GDAL Documentation

Frank Warmerdam, Even Rouault, and others

Jun 05, 2026

CONTENTS

1 What is GDAL? 1

2 License 3
2.1 License . . . 3

3 Download 5
3.1 Source Code . . . 5
3.1.1 Current Release . . . 5
3.1.2 Past Releases . . . 6
3.1.3 Development Source . . . 6
3.1.4 Maintenance policy . . . 6
3.2 Binaries . . . 6
3.2.1 Windows . . . 6
3.2.2 Linux . . . 6
3.2.3 Mac OS . . . 6
3.2.4 Android . . . 6
3.2.5 Cross-Platform Package Managers . . . 7
3.2.5.1 Conda . . . 7
3.2.5.2 pixi . . . 8
3.2.5.3 vcpkg . . . 8
3.2.5.4 Spack . . . 9
3.3 Containers . . . 9
3.4 Documentation . . . 9

4 FAQ 11
4.1 What does GDAL stand for? . . . 11
4.2 What is this OGR stuff? . . . 11
4.3 What does OGR stand for? . . . 11
4.4 What does CPL stand for? . . . 11
4.5 What does VSI stand for? . . . 11
4.6 When was the GDAL project started? . . . 11
4.7 Is GDAL/OGR proprietary software? . . . 11
4.8 What license does GDAL/OGR use? . . . 12
4.9 What operating systems does GDAL-OGR run on? . . . 12
4.10 Is there a graphical user interface to GDAL/OGR? . . . 12
4.10.1 Software using GDAL . . . 12
4.10.1.1 Free and open source . . . 12
4.10.1.2 Proprietary license / Other . . . 14
4.11 What compiler can I use to build GDAL/OGR? . . . 16
4.12 I have a question that's not answered here. Where can I get more information? . . . 16

4.13 How do I add support for a new format? . . . 16
4.14 What about file formats for 3D models? . . . 16
4.15 Is GDAL thread-safe ? . . . 16
4.16 Does GDAL provide a Section 508 information? . . . 16
4.17 How do I cite GDAL ? . . . 16

5 Programs **17**
5.1 "gdal" application . . . 17
5.1.1 General . . . 17
5.1.1.1 Main `gdal` entry point . . . 17
5.1.1.2 Syntax for commands of `gdal` program . . . 19
5.1.1.3 Migration guide to `gdal` command line interface . . . 21
5.1.1.4 Bash completion for `gdal` . . . 25
5.1.1.5 How to use `gdal` CLI algorithms from C . . . 27
5.1.1.6 How to use `gdal` CLI algorithms from C++ . . . 30
5.1.1.7 How to use `gdal` CLI algorithms from Python . . . 31
5.1.1.8 .gdalg files to replay serialized `gdal` commands . . . 35
5.1.2 Commands working with raster or vector inputs . . . 36
5.1.2.1 `gdal info` . . . 36
5.1.2.2 `gdal convert` . . . 64
5.1.2.3 `gdal pipeline` . . . 64
5.1.2.4 `gdal external` . . . 77
5.1.3 Raster commands . . . 79
5.1.3.1 `gdal raster` . . . 79
5.1.3.2 `gdal raster info` . . . 81
5.1.3.3 `gdal raster as-features` . . . 91
5.1.3.4 `gdal raster aspect` . . . 94
5.1.3.5 `gdal raster blend` . . . 97
5.1.3.6 `gdal raster calc` . . . 109
5.1.3.7 `gdal raster clean-collar` . . . 114
5.1.3.8 `gdal raster clip` . . . 116
5.1.3.9 `gdal raster color-map` . . . 121
5.1.3.10 `gdal raster contour` . . . 124
5.1.3.11 `gdal raster compare` . . . 128
5.1.3.12 `gdal raster convert` . . . 130
5.1.3.13 `gdal raster create` . . . 132
5.1.3.14 `gdal raster edit` . . . 136
5.1.3.15 `gdal raster footprint` . . . 141
5.1.3.16 `gdal raster fill-nodata` . . . 146
5.1.3.17 `gdal raster hillshade` . . . 149
5.1.3.18 `gdal raster index` . . . 153
5.1.3.19 `gdal raster materialize` . . . 157
5.1.3.20 `gdal raster mosaic` . . . 159
5.1.3.21 `gdal raster neighbors` . . . 163
5.1.3.22 `gdal raster nodata-to-alpha` . . . 168
5.1.3.23 `gdal raster overview` . . . 170
5.1.3.24 `gdal raster overview add` . . . 170
5.1.3.25 `gdal raster overview delete` . . . 174
5.1.3.26 `gdal raster overview refresh` . . . 175
5.1.3.27 `gdal raster pansharpen` . . . 178
5.1.3.28 `gdal raster pipeline` . . . 181
5.1.3.29 `gdal raster pixel-info` . . . 204
5.1.3.30 `gdal raster polygonize` . . . 211
5.1.3.31 `gdal raster proximity` . . . 214

5.1.3.32 gdal raster pipeline read . . . 217
5.1.3.33 gdal raster reclassify . . . 218
5.1.3.34 gdal raster reproject . . . 222
5.1.3.35 gdal raster resize . . . 229
5.1.3.36 gdal raster rgb-to-palette . . . 231
5.1.3.37 gdal raster roughness . . . 234
5.1.3.38 gdal raster scale . . . 236
5.1.3.39 gdal raster select . . . 240
5.1.3.40 gdal raster set-type . . . 242
5.1.3.41 gdal raster slope . . . 244
5.1.3.42 gdal raster sieve . . . 248
5.1.3.43 gdal raster stack . . . 250
5.1.3.44 gdal raster tile . . . 254
5.1.3.45 gdal raster tpi . . . 261
5.1.3.46 gdal raster tri . . . 263
5.1.3.47 gdal raster unscale . . . 266
5.1.3.48 gdal raster update . . . 268
5.1.3.49 gdal raster viewshed . . . 271
5.1.3.50 gdal raster pipeline write . . . 277
5.1.3.51 gdal raster zonal-stats . . . 278
5.1.4 Vector commands . . . 285
5.1.4.1 gdal vector . . . 285
5.1.4.2 gdal vector buffer . . . 287
5.1.4.3 gdal vector check-coverage . . . 294
5.1.4.4 gdal vector check-geometry . . . 299
5.1.4.5 gdal vector clean-coverage . . . 304
5.1.4.6 gdal vector clip . . . 309
5.1.4.7 gdal vector combine . . . 314
5.1.4.8 gdal vector concat . . . 318
5.1.4.9 gdal vector concave-hull . . . 323
5.1.4.10 gdal vector convert . . . 328
5.1.4.11 gdal vector convex-hull . . . 331
5.1.4.12 gdal vector create . . . 335
5.1.4.13 gdal vector dissolve . . . 339
5.1.4.14 gdal vector edit . . . 342
5.1.4.15 gdal vector filter . . . 346
5.1.4.16 gdal vector info . . . 350
5.1.4.17 gdal vector export-schema . . . 361
5.1.4.18 gdal vector explode-collections . . . 363
5.1.4.19 gdal vector grid . . . 367
5.1.4.20 gdal vector index . . . 375
5.1.4.21 gdal vector layer-algebra . . . 379
5.1.4.22 gdal vector make-point . . . 385
5.1.4.23 gdal vector make-valid . . . 388
5.1.4.24 gdal vector materialize . . . 392
5.1.4.25 gdal vector partition . . . 394
5.1.4.26 gdal vector pipeline . . . 398
5.1.4.27 gdal vector rasterize . . . 421
5.1.4.28 gdal vector pipeline read . . . 426
5.1.4.29 gdal vector rename-layer . . . 427
5.1.4.30 gdal vector reproject . . . 432
5.1.4.31 gdal vector select . . . 436
5.1.4.32 gdal vector segmentize . . . 441
5.1.4.33 gdal vector set-field-type . . . 444

5.1.4.34 gdal vector set-geom-type . . . 448
5.1.4.35 gdal vector sort . . . 453
5.1.4.36 gdal vector simplify . . . 455
5.1.4.37 gdal vector simplify-coverage . . . 460
5.1.4.38 gdal vector sql . . . 464
5.1.4.39 gdal vector swap-xy . . . 469
5.1.4.40 gdal vector update . . . 472
5.1.4.41 gdal vector pipeline write . . . 474
5.1.5 Multidimensional raster commands . . . 477
5.1.5.1 gdal mdim . . . 477
5.1.5.2 gdal mdim info . . . 477
5.1.5.3 gdal mdim convert . . . 482
5.1.5.4 gdal mdim mosaic . . . 485
5.1.6 Dataset management commands . . . 488
5.1.6.1 gdal dataset . . . 488
5.1.6.2 gdal dataset identify . . . 488
5.1.6.3 gdal dataset check . . . 491
5.1.6.4 gdal dataset copy . . . 492
5.1.6.5 gdal dataset rename . . . 493
5.1.6.6 gdal dataset delete . . . 494
5.1.7 Virtual System Interface (VSI) commands . . . 495
5.1.7.1 gdal vsi . . . 495
5.1.7.2 gdal vsi copy . . . 496
5.1.7.3 gdal vsi delete . . . 497
5.1.7.4 gdal vsi list . . . 498
5.1.7.5 gdal vsi move . . . 499
5.1.7.6 gdal vsi sync . . . 500
5.1.7.7 gdal vsi sozip . . . 502
5.1.8 Driver specific commands . . . 507
5.1.8.1 gdal driver cog validate . . . 507
5.1.8.2 gdal driver gpkg repack . . . 508
5.1.8.3 gdal driver gpkg validate . . . 509
5.1.8.4 gdal driver gti create . . . 510
5.1.8.5 gdal driver openfilegdb repack . . . 515
5.1.8.6 gdal driver parquet create-metadata-file . . . 516
5.1.8.7 gdal driver pdf list-layer . . . 517
5.1.8.8 gdal driver zarr add-georeferencing-convention . . . 518
5.2 "Traditional" applications . . . 519
5.2.1 General . . . 519
5.2.1.1 Syntax of arguments of command-line utilities . . . 519
5.2.2 Raster programs . . . 519
5.2.2.1 Common options for raster programs . . . 519
5.2.2.2 gdal-config (Unix) . . . 522
5.2.2.3 gdal2tiles . . . 522
5.2.2.4 gdal2xyz . . . 528
5.2.2.5 gdal_calc . . . 529
5.2.2.6 gdal_contour . . . 534
5.2.2.7 gdal_create . . . 538
5.2.2.8 gdal_edit . . . 540
5.2.2.9 gdal_fillnodata . . . 543
5.2.2.10 gdal_footprint . . . 544
5.2.2.11 gdal_grid . . . 547
5.2.2.12 gdal_merge . . . 554
5.2.2.13 gdal_pansharpen . . . 557

5.2.2.14 gdal_polygonize . . . 559
5.2.2.15 gdal_proximity . . . 561
5.2.2.16 gdal_rasterize . . . 563
5.2.2.17 gdal_retile . . . 566
5.2.2.18 gdal_sieve . . . 569
5.2.2.19 gdal_translate . . . 569
5.2.2.20 gdal_viewshed . . . 577
5.2.2.21 gdaladdo . . . 580
5.2.2.22 gdalattachpct . . . 587
5.2.2.23 gdalbuildvrt . . . 587
5.2.2.24 gdalcompare . . . 592
5.2.2.25 gdaldem . . . 594
5.2.2.26 gdalenhance . . . 602
5.2.2.27 gdalinfo . . . 604
5.2.2.28 gdallocationinfo . . . 614
5.2.2.29 gdalmanage . . . 617
5.2.2.30 gdalmove . . . 619
5.2.2.31 gdalsrsinfo . . . 620
5.2.2.32 gdaltindex . . . 624
5.2.2.33 gdaltransform . . . 629
5.2.2.34 gdalwarp . . . 632
5.2.2.35 nearblack . . . 644
5.2.2.36 pct2rgb . . . 646
5.2.2.37 rgb2pct . . . 647
5.2.3 Multidimensional Raster programs . . . 649
5.2.3.1 gdalmdiminfo . . . 649
5.2.3.2 gdalmdimtranslate . . . 652
5.2.4 Vector programs . . . 655
5.2.4.1 Common options for vector programs . . . 655
5.2.4.2 ogrinfo . . . 656
5.2.4.3 ogr2ogr . . . 665
5.2.4.4 ogrtindex . . . 677
5.2.4.5 ogrlineref . . . 678
5.2.4.6 ogrmerge . . . 681
5.2.4.7 ogr_layer_algebra . . . 684
5.2.5 Geographic network programs . . . 686
5.2.5.1 gnmmanage . . . 686
5.2.5.2 gnmanalyse . . . 688
5.2.6 Other utilities . . . 689
5.2.6.1 sozip . . . 689

6 Raster drivers **693**
6.1 AAIGrid -- Arc/Info ASCII Grid . . . 696
6.1.1 Driver capabilities . . . 697
6.2 ACE2 -- ACE2 . . . 697
6.2.1 Driver capabilities . . . 698
6.3 ADRG -- ADRG/ARC Digitized Raster Graphics (.gen/.thf) . . . 698
6.3.1 Driver capabilities . . . 699
6.4 AIG -- Arc/Info Binary Grid . . . 699
6.4.1 Driver capabilities . . . 699
6.4.1.1 Arc/Info Binary Grid Format . . . 699
6.5 AIRSAR -- AIRSAR Polarimetric Format . . . 710
6.5.1 Driver capabilities . . . 711
6.5.2 See Also . . . 711

6.6 AVIF -- AV1 Image File Format . . . 711
6.6.1 Driver capabilities . . . 712
6.6.2 Color Profile Metadata . . . 712
6.6.3 Georeferencing . . . 712
6.6.4 Creation options . . . 712
6.6.5 See Also . . . 713
6.7 BAG -- Bathymetry Attributed Grid . . . 713
6.7.1 Driver capabilities . . . 713
6.7.2 Open options . . . 714
6.7.3 Variable resolution (VR) grid support . . . 714
6.7.4 Spatial metadata support . . . 716
6.7.5 Tracking list support . . . 716
6.7.6 Creation support . . . 716
6.7.7 Usage examples . . . 717
6.7.8 See Also . . . 719
6.8 BASISU -- Basis Universal . . . 719
6.8.1 Driver capabilities . . . 720
6.8.2 Creation options . . . 720
6.8.3 Build instructions . . . 720
6.9 BMP -- Microsoft Windows Device Independent Bitmap . . . 721
6.9.1 Driver capabilities . . . 721
6.9.2 Creation Options . . . 722
6.9.3 See Also . . . 722
6.10 BSB -- Maptech/NOAA BSB Nautical Chart Format . . . 722
6.10.1 Driver capabilities . . . 722
6.10.2 Metadata . . . 723
6.11 BT -- VTP .bt Binary Terrain Format . . . 723
6.11.1 Driver capabilities . . . 723
6.12 BYN - Natural Resources Canada's Geoid file format (.byn) . . . 724
6.12.1 Driver capabilities . . . 724
6.12.2 Factor . . . 725
6.12.3 See Also . . . 725
6.13 CAD -- AutoCAD DWG raster layer . . . 725
6.13.1 Driver capabilities . . . 725
6.14 CALS -- CALS Type 1 . . . 726
6.14.1 Driver capabilities . . . 726
6.14.2 Metadata . . . 726
6.14.3 Creation issues . . . 727
6.14.4 See Also . . . 727
6.15 CEOS -- CEOS Image . . . 727
6.15.1 Driver capabilities . . . 727
6.16 COASP -- DRDC COASP SAR Processor Raster . . . 727
6.17 COG -- Cloud Optimized GeoTIFF generator . . . 728
6.17.1 Driver capabilities . . . 728
6.17.2 Creation Options . . . 729
6.17.2.1 General creation options . . . 729
6.17.2.2 Reprojection related creation options . . . 733
6.17.3 Update . . . 734
6.17.4 File format details . . . 734
6.17.4.1 High level . . . 734
6.17.4.2 Low level . . . 735
6.17.4.3 Header ghost area . . . 735
6.17.4.4 Tile data leader and trailer . . . 736
6.17.5 Examples . . . 737

6.17.5.1 Example 1: Create a COG with gdalwarp . . . 737
6.17.5.2 Example 2: Create a Web Mercator COG with Google Maps–Compatible Tiling and JPEG Compression . . . 737
6.17.5.3 Example 3: Create a blank GeoTIFF from an existing file . . . 737
6.17.6 See Also . . . 737
6.18 COSAR -- TerraSAR-X Complex SAR Data Product . . . 737
6.18.1 Driver capabilities . . . 738
6.18.2 See Also . . . 738
6.19 CPG -- Convair PolGASP data . . . 738
6.19.1 Driver capabilities . . . 738
6.20 CPHD - Compensated Phase History Data . . . 739
6.20.1 Driver capabilities . . . 739
6.20.2 Open options . . . 739
6.20.3 Multidimensional API support . . . 739
6.20.4 See Also . . . 739
6.21 CTG -- USGS LULC Composite Theme Grid . . . 739
6.21.1 Driver capabilities . . . 740
6.22 DAAS (Airbus DS Intelligence Data As A Service driver) . . . 740
6.22.1 Driver capabilities . . . 741
6.22.2 Dataset name syntax . . . 741
6.22.3 Authentication . . . 741
6.22.4 Configuration options . . . 741
6.22.5 Open options . . . 741
6.23 DDS -- DirectDraw Surface . . . 742
6.23.1 Driver capabilities . . . 743
6.23.2 Build instructions . . . 743
6.23.2.1 Build crunch . . . 743
6.23.2.2 Build GDAL against crunch . . . 743
6.24 DERIVED -- Derived subdatasets driver . . . 743
6.24.1 Available functions . . . 744
6.24.2 Accessing derived subdatasets . . . 744
6.24.3 Listing available subdatasets . . . 746
6.24.4 See Also: . . . 748
6.25 DIMAP -- Spot DIMAP . . . 748
6.25.1 Driver capabilities . . . 748
6.26 DOQ1 -- First Generation USGS DOQ . . . 749
6.26.1 Driver capabilities . . . 749
6.27 DOQ2 -- New Labelled USGS DOQ . . . 749
6.27.1 Driver capabilities . . . 750
6.28 DTED -- Military Elevation Data . . . 750
6.28.1 Driver capabilities . . . 750
6.28.1.1 Configuration options . . . 750
6.28.2 Read Issues . . . 751
6.28.2.1 Read speed . . . 751
6.28.2.2 Georeferencing Issues . . . 751
6.28.2.3 Checksum Issues . . . 751
6.28.3 Creation Issues . . . 751
6.28.4 See Also . . . 752
6.29 E57: ASTM E57 3D file format (image part) . . . 752
6.29.1 Driver capabilities . . . 752
6.29.2 See Also . . . 752
6.30 ECRGTOC -- ECRG Table Of Contents (TOC.xml) . . . 752
6.30.1 Driver capabilities . . . 753
6.30.2 See Also . . . 753

6.31 ECW -- Enhanced Compressed Wavelets (.ecw) . . . 753
6.31.1 Driver capabilities . . . 754
6.31.2 Licensing . . . 754
6.31.3 History . . . 754
6.31.4 Creation Options . . . 754
6.31.5 Creation Options: . . . 755
6.31.6 Configuration Options . . . 755
6.31.6.1 ECW Version 3 Files . . . 756
6.31.6.2 ECWP . . . 756
6.31.6.3 Metadata / Georeferencing . . . 756
6.31.6.4 File Metadata Keys: . . . 756
6.31.7 Standalone plugin compilation . . . 757
6.31.8 See Also . . . 757
6.32 EEDAI - Google Earth Engine Data API Image . . . 758
6.32.1 Driver capabilities . . . 758
6.32.2 Dataset name syntax . . . 758
6.32.3 Open options . . . 758
6.32.4 Authentication methods . . . 758
6.32.5 Configuration options . . . 759
6.32.6 Overviews . . . 759
6.32.7 Subdatasets . . . 759
6.32.8 Metadata . . . 759
6.32.9 Pixel encoding . . . 759
6.32.9.1 Examples . . . 760
6.32.10 See Also . . . 760
6.33 EHdr -- ESRI .hdr Labelled . . . 760
6.33.1 Driver capabilities . . . 761
6.33.2 See Also . . . 761
6.34 EIR -- Erdas Imagine Raw . . . 761
6.34.1 Driver capabilities . . . 762
6.35 ENVI -- ENVI .hdr Labelled Raster . . . 762
6.35.1 Driver capabilities . . . 763
6.36 ESAT -- Envisat Image Product . . . 763
6.36.1 Driver capabilities . . . 764
6.37 ESRIC -- Esri Compact Cache . . . 764
6.37.1 Driver capabilities . . . 764
6.37.2 Esri Compact Cache V2 . . . 764
6.37.3 Esri Tile Package (.tpkx) . . . 764
6.37.3.1 Usage examples . . . 765
6.37.3.2 Features and Limitations . . . 765
6.37.4 Open options . . . 766
6.37.5 See Also . . . 766
6.38 ERS -- ERMapper .ERS . . . 766
6.38.1 Driver capabilities . . . 766
6.38.2 Creation Options . . . 767
6.38.3 See Also . . . 767
6.39 EXR -- Extended Dynamic Range Image File Format . . . 767
6.39.1 Creation Options . . . 768
6.39.2 Driver capabilities . . . 768
6.40 FAST -- EOSAT FAST Format . . . 769
6.40.1 Driver capabilities . . . 769
6.40.2 Data . . . 770
6.40.2.1 FAST-L7A . . . 770
6.40.2.2 IRS-1C/1D . . . 770

6.40.3 Georeference . . . 771
6.40.4 Metadata . . . 771
6.40.5 See Also . . . 771
6.41 FITS -- Flexible Image Transport System . . . 771
6.41.1 Multiple image support . . . 772
6.41.2 Binary table support . . . 772
6.41.3 Layer creation options . . . 773
6.41.4 Examples . . . 773
6.41.5 Other . . . 774
6.41.6 Notes on CFITSIO linking in GDAL . . . 774
6.41.6.1 Linux . . . 774
6.41.6.2 MacOSX . . . 774
6.41.7 Driver capabilities . . . 774
6.42 GenBin -- Generic Binary (.hdr labelled) . . . 775
6.42.1 Driver capabilities . . . 775
6.43 Oracle Spatial GeoRaster . . . 776
6.43.1 Driver capabilities . . . 777
6.43.2 Browsing the database for GeoRasters . . . 777
6.43.3 Creation Options . . . 778
6.43.4 Open Options . . . 780
6.43.5 Importing GeoRaster . . . 780
6.43.6 Exporting GeoRaster . . . 781
6.43.7 Cross schema access . . . 781
6.43.8 General use of GeoRaster . . . 782
6.43.8.1 Standalone plugin compilation . . . 782
6.44 GFF -- Sandia National Laboratories GSAT File Format . . . 782
6.44.1 Driver capabilities . . . 783
6.45 GIF -- Graphics Interchange Format . . . 783
6.45.1 Driver capabilities . . . 783
6.45.2 Creation Issues . . . 784
6.45.3 See Also . . . 784
6.46 GPKG -- GeoPackage raster . . . 784
6.46.1 Driver capabilities . . . 785
6.46.2 Opening options . . . 785
6.46.3 Creation issues . . . 786
6.46.3.1 Tile formats . . . 787
6.46.3.2 Tiling schemes . . . 787
6.46.3.3 Nodata value . . . 788
6.46.3.4 Creation options . . . 789
6.46.4 Overviews . . . 791
6.46.5 Metadata . . . 791
6.46.6 IMAGE_STRUCTURE metadata item . . . 791
6.46.7 Level of support of GeoPackage Extensions . . . 792
6.46.8 Examples . . . 792
6.46.8.1 Raster SQL functions . . . 793
6.46.9 See Also . . . 793
6.46.10 Other notes . . . 793
6.47 GRASS Raster Format . . . 794
6.47.1 See Also . . . 794
6.48 GRASSASCIIGrid -- GRASS ASCII Grid . . . 794
6.48.1 Driver capabilities . . . 794
6.49 GRIB -- WMO General Regularly-distributed Information in Binary form . . . 795
6.49.1 Driver capabilities . . . 796
6.49.2 Configuration options . . . 796

6.49.3 Open options . . . 796
6.49.4 GRIB2 write support . . . 797
6.49.4.1 Product identification and definition . . . 797
6.49.4.2 Data encoding . . . 798
6.49.4.3 Data units . . . 799
6.49.4.4 GRIB2 to GRIB2 conversions . . . 799
6.49.4.5 Examples . . . 799
6.49.5 See Also: . . . 799
6.49.6 Credits . . . 800
6.50 GS7BG -- Golden Software Surfer 7 Binary Grid File Format . . . 800
6.50.1 Driver capabilities . . . 800
6.51 GSAG -- Golden Software ASCII Grid File Format . . . 801
6.51.1 Driver capabilities . . . 801
6.52 GSBG -- Golden Software Binary Grid File Format . . . 801
6.52.1 Driver capabilities . . . 802
6.53 GSC -- GSC Geogrid . . . 802
6.53.1 Driver capabilities . . . 802
6.54 GDALG: GDAL Streamed Algorithm . . . 802
6.54.1 Driver capabilities . . . 804
6.55 GTA - Generic Tagged Arrays . . . 804
6.55.1 Driver capabilities . . . 804
6.55.2 Creation options . . . 804
6.55.3 See Also . . . 804
6.56 GTI -- GDAL Raster Tile Index . . . 805
6.56.1 Introduction . . . 805
6.56.2 Connection strings . . . 805
6.56.3 STAC GeoParquet support . . . 806
6.56.4 Tile index requirements . . . 806
6.56.5 GTI XML format . . . 809
6.56.6 How to build a GTI compatible index ? . . . 811
6.56.7 Open options . . . 811
6.56.8 Overviews . . . 812
6.56.8.1 Overviews in XML . . . 812
6.56.8.2 Overviews as metadata items of a GDAL vector layer . . . 814
6.56.9 Multi-threading optimizations . . . 815
6.57 GTiff -- GeoTIFF File Format . . . 815
6.57.1 Driver capabilities . . . 816
6.57.2 Georeferencing . . . 816
6.57.3 Internal nodata masks . . . 817
6.57.4 Overviews . . . 817
6.57.5 Overviews and nodata masks . . . 819
6.57.6 Metadata . . . 820
6.57.7 Color Profile Metadata . . . 820
6.57.8 json:ISIS3 Metadata . . . 821
6.57.9 Nodata value . . . 821
6.57.10 Raster Attribute Table . . . 821
6.57.11 Sparse files . . . 821
6.57.12 Raw mode . . . 822
6.57.13 Open options . . . 822
6.57.14 Creation Issues . . . 822
6.57.14.1 Creation Options . . . 823
6.57.14.2 Subdatasets . . . 827
6.57.14.3 About JPEG compression of RGB images . . . 828
6.57.14.4 Lossless conversion of JPEG into JPEG-in-TIFF . . . 828

6.57.14.5 Streaming operations . . . 828
6.57.14.6 Configuration options . . . 829
6.57.15 Codec Recommendations . . . 831
6.57.15.1 LZW . . . 831
6.57.15.2 DEFLATE . . . 831
6.57.15.3 LERC . . . 831
6.57.15.4 ZSTD . . . 831
6.57.15.5 WEBP . . . 832
6.57.15.6 JXL . . . 832
6.57.15.7 LZMA . . . 832
6.57.15.8 CCITTRLE/CCITTFAX3/CCITTFAX4 . . . 832
6.57.16 See Also . . . 832
6.57.16.1 WLD -- ESRI World File . . . 832
6.58 GXF -- Grid eXchange File . . . 832
6.58.1 Driver capabilities . . . 833
6.59 GTX — NOAA Vertical Datum Grid Shift . . . 833
6.59.1 Driver capabilities . . . 833
6.59.2 File format . . . 834
6.59.3 See also . . . 834
6.60 HDF4 -- Hierarchical Data Format Release 4 (HDF4) . . . 834
6.60.1 Driver capabilities . . . 835
6.60.2 Multiple Image Handling (Subdatasets) . . . 835
6.60.3 Georeference . . . 837
6.60.4 Creation Issues . . . 837
6.60.5 Metadata . . . 838
6.60.6 Open options . . . 838
6.60.7 Multidimensional API support . . . 838
6.60.8 Driver building . . . 838
6.60.9 See Also . . . 838
6.61 HDF5 -- Hierarchical Data Format Release 5 (HDF5) . . . 839
6.61.1 Driver capabilities . . . 839
6.61.2 Multiple Image Handling (Subdatasets) . . . 839
6.61.3 Georeference . . . 843
6.61.4 Multi-file support . . . 844
6.61.5 Multidimensional API support . . . 844
6.61.6 Driver building . . . 844
6.61.7 See Also . . . 844
6.62 HEIF / HEIC -- ISO/IEC 23008-12 High Efficiency Image File Format . . . 844
6.62.1 AVIF support . . . 845
6.62.2 Driver capabilities . . . 845
6.62.3 Georeferencing . . . 845
6.62.4 Built hints on Windows . . . 845
6.63 HF2 -- HF2/HFZ heightfield raster . . . 846
6.63.1 Driver capabilities . . . 846
6.63.2 Creation options . . . 847
6.63.3 See also . . . 847
6.64 HFA -- Erdas Imagine .img . . . 847
6.64.1 Driver capabilities . . . 847
6.64.2 Creation Issues . . . 848
6.64.3 Configuration Options . . . 849
6.64.4 See Also . . . 849
6.65 RST -- Idrisi Raster Format . . . 849
6.65.1 Driver capabilities . . . 850
6.65.2 See Also . . . 850

6.66 ILWIS -- Raster Map . . . 851
6.66.1 Driver capabilities . . . 851
6.67 IRIS -- Vaisala's weather radar software format . . . 852
6.67.1 Driver capabilities . . . 852
6.68 ISCE -- ISCE . . . 853
6.68.1 Driver capabilities . . . 853
6.69 ISG -- International Service for the Geoid . . . 853
6.69.1 Driver capabilities . . . 854
6.70 ISIS2 -- USGS Astrogeology ISIS Cube (Version 2) . . . 854
6.70.1 Driver capabilities . . . 855
6.70.2 See Also . . . 855
6.71 ISIS3 -- USGS Astrogeology ISIS Cube (Version 3) . . . 855
6.71.1 Driver capabilities . . . 856
6.71.2 Metadata . . . 856
6.71.3 Creation support . . . 858
6.71.4 Open options . . . 859
6.71.5 Examples . . . 859
6.71.6 See Also . . . 860
6.72 JDEM -- Japanese DEM (.mem) . . . 860
6.72.1 Driver capabilities . . . 861
6.73 JP2ECW -- ERDAS JPEG2000 (.jp2) . . . 861
6.73.1 Driver capabilities . . . 861
6.73.2 Licensing . . . 862
6.73.3 History . . . 862
6.73.4 Georeferencing . . . 862
6.73.5 Open Options . . . 862
6.73.6 Creation Options: . . . 862
6.73.7 Create support . . . 864
6.73.8 Configuration Options . . . 864
6.73.9 Metadata . . . 865
6.73.10 See Also . . . 865
6.74 JP2Grok -- JPEG2000 driver based on Grok library . . . 866
6.74.1 Driver capabilities . . . 866
6.74.2 Georeferencing . . . 866
6.74.3 Thread support . . . 867
6.74.4 Open Options . . . 867
6.74.5 Creation Options . . . 867
6.74.5.1 Lossless compression . . . 870
6.74.6 Standalone plugin compilation . . . 870
6.74.7 See Also . . . 871
6.75 JP2KAK -- JPEG 2000 (based on Kakadu SDK) . . . 871
6.75.1 Driver capabilities . . . 872
6.75.2 Configuration Options . . . 872
6.75.3 Georeferencing . . . 872
6.75.4 Option Options . . . 873
6.75.5 Creation Issues . . . 873
6.75.6 Known Kakadu SDK Issues . . . 874
6.75.6.1 Alpha Channel Writing in v7.8 . . . 874
6.75.6.2 kdu_get_num_processors always returns 0 for some platforms . . . 874
6.75.7 Standalone plugin compilation . . . 875
6.75.8 See Also . . . 875
6.76 JP2MrSID -- JPEG2000 via MrSID SDK . . . 875
6.76.1 Driver capabilities . . . 875
6.76.2 Georeferencing . . . 876

6.76.3 Open Options . . . 876
6.76.4 Creation Options . . . 876
6.76.5 See Also . . . 877
6.77 JP2OpenJPEG -- JPEG2000 driver based on OpenJPEG library . . . 877
6.77.1 Driver capabilities . . . 878
6.77.2 Georeferencing . . . 878
6.77.3 Thread support . . . 878
6.77.4 Open Options . . . 879
6.77.5 Creation Options . . . 879
6.77.5.1 Lossless compression . . . 882
6.77.5.2 GMLJP2v2 definition file . . . 882
6.77.6 Vector information . . . 885
6.77.7 Standalone plugin compilation . . . 886
6.77.8 See Also . . . 886
6.78 JPEG -- JPEG JFIF File Format . . . 886
6.78.1 Driver capabilities . . . 887
6.78.2 Color Profile Metadata . . . 887
6.78.3 Configuration options . . . 888
6.78.4 Open Options . . . 888
6.78.5 Creation Options . . . 888
6.78.6 EXIF and GPS tags . . . 889
6.78.7 FLIR metadata . . . 892
6.78.8 See Also . . . 893
6.79 JPEGXL -- JPEG-XL File Format . . . 893
6.79.1 Driver capabilities . . . 893
6.79.2 Color Profile Metadata . . . 894
6.79.3 Open Options . . . 894
6.79.4 Creation Options . . . 894
6.79.5 See Also . . . 895
6.80 JPIPKAK - JPIP Streaming . . . 895
6.80.1 Driver capabilities . . . 895
6.80.2 JPIPKAK - JPIP Overview . . . 895
6.80.3 JPIPKAK -approach . . . 896
6.80.4 JPIPKAK - implementation . . . 897
6.80.5 JPIPKAK - installation requirements . . . 900
6.80.6 See Also . . . 901
6.80.7 NOTES . . . 901
6.81 KEA . . . 901
6.81.1 Driver capabilities . . . 902
6.81.2 Creation options . . . 902
6.81.3 See Also . . . 903
6.82 KMLSuperoverlay -- KMLSuperoverlay . . . 903
6.82.1 Driver capabilities . . . 903
6.83 KRO -- KOLOR Raw format . . . 903
6.83.1 Driver capabilities . . . 904
6.84 KTX2 . . . 904
6.84.1 Driver capabilities . . . 905
6.84.2 Creation options . . . 905
6.84.3 Build instructions . . . 906
6.85 LAN -- Erdas 7.x .LAN and .GIS . . . 906
6.85.1 Driver capabilities . . . 906
6.86 L1B -- NOAA Polar Orbiter Level 1b Data Set (AVHRR) . . . 907
6.86.1 Driver capabilities . . . 907
6.86.2 Georeference . . . 907

6.86.3 Data . . . 908
6.86.4 Metadata . . . 908
6.86.5 Subdatasets . . . 908
6.86.6 Nodata mask . . . 908
6.86.7 See Also . . . 909
6.87 LCP -- FARSITE v.4 LCP Format . . . 909
6.87.1 Driver capabilities . . . 909
6.87.2 Metadata . . . 910
6.87.2.1 Dataset . . . 910
6.87.2.2 Band . . . 910
6.87.3 Creation Options . . . 910
6.88 Leveller -- Daylon Leveller Heightfield . . . 913
6.88.1 Driver capabilities . . . 913
6.88.2 Reading . . . 913
6.88.3 Writing . . . 914
6.88.4 See Also: . . . 914
6.89 LIBERTIFF -- GeoTIFF File Format . . . 914
6.89.1 Driver capabilities . . . 914
6.89.2 Open options . . . 915
6.90 LOSLAS -- NADCON .los/.las Datum Grid Shift . . . 915
6.90.1 Driver capabilities . . . 915
6.91 MAP -- OziExplorer .MAP . . . 916
6.91.1 Driver capabilities . . . 916
6.91.2 See Also: . . . 916
6.92 MRF -- Meta Raster Format . . . 916
6.92.1 Driver capabilities . . . 917
6.92.2 Links . . . 917
6.93 MBTiles . . . 917
6.93.1 Driver capabilities . . . 918
6.93.2 Configuration options . . . 918
6.93.3 Opening options . . . 918
6.93.4 Raster creation issues . . . 919
6.93.4.1 Tile formats . . . 920
6.93.5 Vector creation issues . . . 920
6.93.6 Creation options . . . 920
6.93.7 Layer configuration (vector) . . . 921
6.93.8 Layer creation options (vector) . . . 922
6.93.9 Overviews (raster) . . . 922
6.93.10 Vector tiles . . . 922
6.93.11 Examples: . . . 923
6.93.12 See Also . . . 925
6.94 MEM -- In Memory datasets . . . 925
6.94.1 Dataset Name Format . . . 925
6.94.2 Creation Options . . . 926
6.94.3 Driver capabilities . . . 926
6.94.4 Multidimensional API support . . . 927
6.95 MFF -- Vexcel MFF Raster . . . 927
6.95.1 Driver capabilities . . . 927
6.96 MFF2 -- Vexcel MFF2 Image . . . 928
6.96.1 Driver capabilities . . . 928
6.96.2 Format Details . . . 928
6.96.2.1 MFF2 Top-level Structure . . . 928
6.96.2.2 The "attrib" File . . . 928
6.96.2.3 The "image_data" File . . . 929

6.96.2.4 The "georef" File . . . 929
6.96.2.5 Supported projections . . . 930
6.96.2.6 Recognized ellipsoids . . . 930
6.96.2.7 Explanation of fields . . . 931
6.97 MiraMon Raster . . . 931
6.97.1 Driver capabilities . . . 932
6.97.2 Overview of MiraMon format . . . 932
6.97.3 Writing behavior . . . 933
6.97.4 Color Table . . . 933
6.97.5 Attribute Table . . . 933
6.97.6 Metadata . . . 933
6.97.7 Encoding . . . 933
6.97.8 Subdataset Generation . . . 933
6.97.9 Open options . . . 934
6.97.10 Creation Options . . . 934
6.97.11 Open examples . . . 934
6.97.12 Creation examples . . . 935
6.97.13 See Also . . . 935
6.98 MrSID -- Multi-resolution Seamless Image Database . . . 936
6.98.1 Driver capabilities . . . 936
6.98.2 Metadata . . . 936
6.98.3 Georeference . . . 937
6.98.4 Standalone plugin compilation . . . 937
6.98.5 See Also: . . . 937
6.99 MSG -- Meteosat Second Generation . . . 937
6.99.1 Driver capabilities . . . 938
6.99.2 Build Instructions . . . 938
6.99.3 Specification of Source Dataset . . . 938
6.99.4 Georeference and Projection . . . 940
6.99.5 See Also . . . 940
6.100 MSGN -- Meteosat Second Generation (MSG) Native Archive Format (.nat) . . . 940
6.100.1 Driver capabilities . . . 940
6.101 NDF -- NLAPS Data Format . . . 941
6.101.1 Driver capabilities . . . 941
6.102 NetCDF: Network Common Data Form . . . 941
6.102.1 Driver capabilities . . . 942
6.102.2 Multiple Image Handling (Subdatasets) . . . 942
6.102.3 Dimension Ordering . . . 945
6.102.4 Georeferencing . . . 946
6.102.4.1 Spatial reference system . . . 946
6.102.4.2 Geotransform . . . 946
6.102.5 Open options . . . 947
6.102.6 Creation Issues . . . 947
6.102.7 GDAL NetCDF Metadata . . . 947
6.102.8 Product specific behavior . . . 948
6.102.8.1 Sentinel 5 . . . 948
6.102.9 Creation Options . . . 948
6.102.10 Creation of multidimensional files with CreateCopy() 2D raster API . . . 949
6.102.11 Configuration Options . . . 950
6.102.12 VSI Virtual File System API support . . . 951
6.102.13 NetCDF-4 groups support on reading (GDAL >= 3.0) . . . 951
6.102.14 Multidimensional API support . . . 951
6.102.15 Driver building . . . 952
6.102.16 See Also: . . . 952

6.103 NGSGEOID - NOAA NGS Geoid Height Grids . . . 953
6.103.1 Driver capabilities . . . 953
6.103.2 See also . . . 953
6.104 NGW -- NextGIS Web . . . 953
6.104.1 Driver capabilities . . . 954
6.104.2 Driver . . . 954
6.104.3 Dataset name syntax . . . 954
6.104.4 Configuration options . . . 955
6.104.5 Authentication . . . 955
6.104.6 Open options . . . 955
6.104.7 Create copy options . . . 956
6.104.8 Metadata . . . 956
6.104.9 Examples . . . 956
6.104.10 See also . . . 957
6.105 NITF -- National Imagery Transmission Format (also CIB, CADRG, ECRG, HRE) . . . 957
6.105.1 NITF -- Advanced Driver Information . . . 957
6.105.1.1 CGM Segments . . . 957
6.105.1.2 Multi-Image NITF Files . . . 958
6.105.1.3 Text Segments . . . 958
6.105.1.4 TREs . . . 959
6.105.1.5 TREs as xml:TRE . . . 959
6.105.1.6 TRE creation from hexadecimal data . . . 960
6.105.1.7 TRE creation as TRE_OVERFLOW DES . . . 960
6.105.1.8 Data Extension Segments (xml:DES) . . . 961
6.105.1.9 Raw File / Image Headers . . . 961
6.105.1.10 Writing multiple Image Segments . . . 961
6.105.1.11 Reading SAR products . . . 962
6.105.2 Driver capabilities . . . 962
6.105.3 Open options . . . 963
6.105.4 Creation Issues . . . 963
6.105.5 JPEG2000 compression (write support) . . . 969
6.105.6 CADRG (Compressed ARC Digitized Raster Graphics) (write support) . . . 970
6.105.6.1 Example 1: Create CADRG frames from a VRT mosaic, specifying it is an Operational Navigation Chart (scale 1:1 million), using hard-coded dates . . . 970
6.105.6.2 Example 2: Create CADRG frames from a VRT mosaic, specifying it is a City Graphic map at scale 1:5,000, with dates corresponding to the current timestamp. . . . 970
6.105.7 Links . . . 970
6.105.8 Credit . . . 971
6.106 NOAA_B -- NOAA GEOCON/NADCON5 .b format . . . 971
6.106.1 Driver capabilities . . . 971
6.106.2 See also . . . 971
6.107 NSIDCbin -- National Snow and Ice Data Centre Sea Ice Concentrations . . . 971
6.107.1 Driver capabilities . . . 972
6.107.2 Example . . . 972
6.108 NTv2 -- NTv2 Datum Grid Shift . . . 972
6.108.1 Driver capabilities . . . 973
6.109 NWT_GRD/NWT_GRC -- Northwood/Vertical Mapper File Format . . . 973
6.109.1 Driver capabilities (NWT_GRD) . . . 973
6.109.2 Driver capabilities (NWT_GRC) . . . 974
6.109.3 Color Information . . . 974
6.109.4 Metadata . . . 974
6.109.5 Nodata values . . . 974
6.109.5.1 Creation Options . . . 974
6.110 OGCAPI -- OGC API Tiles / Maps / Coverage . . . 975

6.110.1 Driver capabilities . . . 975
6.110.2 Dataset opening . . . 975
6.110.3 Open options . . . 976
6.111 ESRI File Geodatabase raster (OpenFileGDB) . . . 977
6.111.1 Driver capabilities . . . 977
6.111.2 Connection string . . . 978
6.111.3 Open options . . . 978
6.111.4 Metadata . . . 978
6.111.5 Examples . . . 978
6.111.6 Links . . . 978
6.111.7 Credits . . . 978
6.112 JAXA PALSAR Processed Products . . . 979
6.112.1 Driver capabilities . . . 979
6.112.2 See Also . . . 979
6.113 PAux -- PCI .aux Labelled Raw Format . . . 980
6.113.1 Driver capabilities . . . 980
6.114 PCIDSK -- PCI Geomatics Database File . . . 980
6.114.1 Creation Options . . . 981
6.114.2 Driver capabilities . . . 981
6.114.3 See Also: . . . 981
6.115 PCRaster -- PCRaster raster file format . . . 981
6.115.1 Driver capabilities . . . 982
6.116 PDF -- Geospatial PDF . . . 983
6.116.1 Driver capabilities . . . 983
6.116.2 Vector support . . . 984
6.116.3 Metadata . . . 984
6.116.4 Configuration options . . . 984
6.116.4.1 Open Options . . . 985
6.116.5 LAYERS Metadata domain . . . 985
6.116.6 Restrictions . . . 985
6.116.7 Creation Issues . . . 986
6.116.7.1 Creation Options . . . 986
6.116.8 Update of existing files . . . 988
6.116.9 Creation of PDF file from a XML composition file (GDAL >= 3.0) . . . 988
6.116.10 Build dependencies . . . 991
6.116.10.1 Poppler . . . 991
6.116.10.2 PoDoFo . . . 991
6.116.10.3 PDFium . . . 991
6.116.11 Examples . . . 993
6.116.12 See also . . . 993
6.117 PDS -- Planetary Data System v3 . . . 993
6.117.1 Driver capabilities . . . 994
6.117.2 Configuration options . . . 994
6.117.3 Georeferencing . . . 994
6.117.4 See Also . . . 995
6.118 PDS4 -- NASA Planetary Data System (Version 4) . . . 995
6.118.1 Driver capabilities . . . 996
6.118.2 Metadata . . . 996
6.118.3 Open options (vector only) . . . 996
6.118.4 Creation support . . . 997
6.118.5 Layer creation options (vector/table datasets) . . . 998
6.118.6 Subdataset / multiple image support . . . 998
6.118.7 PDS4 raster examples . . . 999
6.118.8 PDS4 vector examples . . . 1001

6.118.9 See Also: . . . 1002
6.119 PLMosaic (Planet Labs Mosaics API) . . . 1002
6.119.1 Driver capabilities . . . 1002
6.119.2 Dataset name syntax . . . 1002
6.119.3 Open options . . . 1003
6.119.4 Configuration options . . . 1003
6.119.5 Location information . . . 1003
6.119.5.1 Examples . . . 1004
6.119.6 See Also . . . 1006
6.120 PMTiles . . . 1006
6.120.1 Driver capabilities . . . 1006
6.120.2 Opening options . . . 1007
6.120.3 Creation issues . . . 1007
6.120.4 Creation options . . . 1007
6.120.5 /vsipmtiles/ virtual file system . . . 1008
6.120.6 Examples . . . 1008
6.120.6.1 Example 1: Simple translation of a GeoTIFF into PMTiles using default PNG tile format . . . 1008
6.120.7 See Also . . . 1008
6.121 PNG -- Portable Network Graphics . . . 1008
6.121.1 Driver capabilities . . . 1009
6.121.2 Transparency color . . . 1009
6.121.3 Color Profile Metadata . . . 1009
6.122 PNM -- Netpbm (.pgm, .ppm) . . . 1010
6.122.1 Creation Options . . . 1010
6.122.2 Driver capabilities . . . 1010
6.123 PostGISRaster -- PostGIS Raster driver . . . 1011
6.123.1 Driver capabilities . . . 1011
6.123.2 Connecting to a database . . . 1011
6.123.2.1 Additional notes . . . 1012
6.123.2.2 Performance hints . . . 1012
6.123.3 Examples . . . 1012
6.123.4 Credits . . . 1012
6.123.5 See Also . . . 1013
6.124 PHOTOMOD Raster File . . . 1013
6.124.1 Driver capabilities . . . 1013
6.124.2 See Also . . . 1013
6.125 RasterLite2 - Rasters in SQLite DB . . . 1014
6.125.1 Driver capabilities . . . 1014
6.125.2 Opening syntax . . . 1014
6.125.3 Creation . . . 1015
6.125.4 Creation options . . . 1015
6.125.5 Examples . . . 1015
6.125.6 See Also . . . 1016
6.126 RCM -- RADARSAT Constellation Mission Product . . . 1016
6.126.1 Driver capabilities . . . 1016
6.126.2 Data Calibration . . . 1017
6.126.3 See Also . . . 1017
6.127 RIK -- Swedish Grid Maps . . . 1017
6.127.1 Driver capabilities . . . 1017
6.128 RMF -- Raster Matrix Format . . . 1018
6.128.1 Driver capabilities . . . 1018
6.128.2 Metadata . . . 1018
6.128.3 Open Options . . . 1019

6.128.4 Creation Options . . . 1019
6.128.5 See Also: . . . 1019
6.129 ROI_PAC -- ROI_PAC . . . 1019
6.129.1 Driver capabilities . . . 1020
6.130 RPFTOC -- Raster Product Format/RPF (a.toc) . . . 1020
6.130.1 Driver capabilities . . . 1021
6.130.2 Open options . . . 1021
6.130.3 Configuration options . . . 1021
6.130.4 Creation of a A.TOC file from existing CADRG frames . . . 1022
6.130.4.1 Description . . . 1022
6.130.4.2 Synopsis . . . 1022
6.130.4.3 Program-Specific Options . . . 1022
6.130.4.4 Examples . . . 1023
6.131 RRASTER -- R Raster . . . 1023
6.131.1 Driver capabilities . . . 1023
6.131.2 See Also . . . 1024
6.132 RS2 -- RadarSat 2 XML Product . . . 1024
6.132.1 Driver capabilities . . . 1024
6.132.2 Data Calibration . . . 1025
6.132.3 See Also . . . 1025
6.133 S102 -- S-102 Bathymetric Surface Product . . . 1025
6.133.1 Driver capabilities . . . 1026
6.133.2 Open options . . . 1026
6.133.3 Spatial metadata support . . . 1026
6.133.4 Write support . . . 1026
6.133.5 Validation script . . . 1028
6.133.6 Examples . . . 1029
6.133.7 See Also . . . 1029
6.134 S104 -- S-104 Water Level Information for Surface Navigation Product . . . 1029
6.134.1 Driver capabilities . . . 1030
6.134.2 Open options . . . 1030
6.134.3 Write support . . . 1030
6.134.4 Validation script . . . 1032
6.134.5 Examples . . . 1033
6.134.6 See Also . . . 1033
6.135 S111 -- S-111 Surface Currents Product . . . 1033
6.135.1 Driver capabilities . . . 1034
6.135.2 Open options . . . 1034
6.135.3 Write support . . . 1034
6.135.4 Validation script . . . 1037
6.135.5 Examples . . . 1037
6.135.6 See Also . . . 1037
6.136 SAFE -- Sentinel-1 SAFE XML Product . . . 1037
6.136.1 Driver capabilities . . . 1038
6.136.2 Multiple measurements . . . 1038
6.136.3 Examples . . . 1038
6.136.4 Data Calibration . . . 1040
6.136.5 See Also . . . 1040
6.137 SAR_CEOS -- CEOS SAR Image . . . 1040
6.137.1 Driver capabilities . . . 1041
6.138 SAGA -- SAGA GIS Binary Grid File Format . . . 1041
6.138.1 Driver capabilities . . . 1042
6.139 SENTINEL2 -- Sentinel-2 Products . . . 1042
6.139.1 Driver capabilities . . . 1043

6.139.2 Level-1B . . . 1043
6.139.2.1 Default / per-granule mode (L1B_MODE = GRANULE) . . . 1043
6.139.2.2 Per-datastrip per-detector per-band mode (L1B_MODE = DATASTRIP) . . . 1044
6.139.3 Level-1C . . . 1044
6.139.4 Level-2A . . . 1044
6.139.5 Metadata . . . 1045
6.139.6 Performance issues for L1C and L2A . . . 1045
6.139.7 Open options . . . 1045
6.139.8 Examples . . . 1045
6.139.9 See Also . . . 1051
6.139.10 Credits . . . 1051
6.140 SIGDEM -- Scaled Integer Gridded DEM . . . 1051
6.140.1 Driver capabilities . . . 1051
6.141 SNAP_TIFF -- Sentinel Application Processing GeoTIFF . . . 1052
6.141.1 Driver capabilities . . . 1052
6.141.2 Georeferencing . . . 1052
6.141.3 Metadata . . . 1052
6.142 SNODAS -- Snow Data Assimilation System . . . 1052
6.142.1 Driver capabilities . . . 1053
6.143 SRP -- Standard Product Format (ASRP/USRP) (.gen) . . . 1053
6.143.1 Driver capabilities . . . 1054
6.144 SRTMHGT -- SRTM HGT Format . . . 1054
6.144.1 Driver capabilities . . . 1054
6.145 STACIT - Spatio-Temporal Asset Catalog Items . . . 1055
6.145.1 Open syntax . . . 1055
6.145.2 Open options . . . 1056
6.145.3 Subdatasets . . . 1056
6.145.4 Driver capabilities . . . 1056
6.145.5 Examples . . . 1056
6.145.6 See Also . . . 1058
6.146 STACTA - Spatio-Temporal Asset Catalog Tiled Assets . . . 1058
6.146.1 Configuration options . . . 1058
6.146.2 Open syntax . . . 1058
6.146.3 Open options . . . 1058
6.146.4 Subdatasets . . . 1059
6.146.5 Driver capabilities . . . 1059
6.146.6 Creation . . . 1059
6.146.7 See Also . . . 1059
6.147 Terragen -- Terragen™ Terrain File . . . 1059
6.147.1 Driver capabilities . . . 1059
6.147.2 Reading . . . 1060
6.147.3 Writing . . . 1060
6.147.4 Roundtripping . . . 1060
6.147.5 See Also . . . 1061
6.148 TGA -- TARGA Image File Format . . . 1061
6.148.1 Driver capabilities . . . 1061
6.148.2 Links . . . 1061
6.149 TIL -- EarthWatch/DigitalGlobe .TIL . . . 1061
6.149.1 Driver capabilities . . . 1062
6.150 TileDB - TileDB raster . . . 1062
6.150.1 Driver capabilities . . . 1062
6.150.2 Open options . . . 1063
6.150.3 Creation options . . . 1063
6.150.4 Multidimensional API support . . . 1063

6.150.5 Standalone plugin compilation . . . 1064
6.150.6 See Also . . . 1064
6.151 TSX -- TerraSAR-X Product . . . 1065
6.151.1 Driver capabilities . . . 1065
6.152 USGSDEM -- USGS ASCII DEM (and CDED) . . . 1065
6.152.1 Driver capabilities . . . 1066
6.153 VICAR -- VICAR . . . 1066
6.153.1 Driver capabilities . . . 1066
6.153.2 Metadata . . . 1067
6.153.3 Binary prefixes . . . 1069
6.153.4 Creation support . . . 1069
6.153.5 See Also . . . 1070
6.154 VRT -- GDAL Virtual Format . . . 1070
6.154.1 Introduction . . . 1070
6.154.2 Creation options . . . 1071
6.154.3 .vrt Format . . . 1071
6.154.3.1 VRTDataset . . . 1072
6.154.3.2 VRTRasterBand . . . 1073
6.154.4 Overviews . . . 1084
6.154.5 .vrt Descriptions for Raw Files . . . 1084
6.154.6 Restricting access to Raw Files . . . 1085
6.154.7 Creation of VRT Datasets . . . 1086
6.154.8 Derived Bands (pixel functions) . . . 1089
6.154.8.1 Example 1: Calculating a derived band . . . 1089
6.154.8.2 Built-in Pixel Functions . . . 1090
6.154.8.3 Writing Pixel Functions in C/C++ . . . 1095
6.154.8.4 Writing Pixel Functions in Python . . . 1098
6.154.8.5 Python pixel function examples . . . 1099
6.154.8.6 Security implications . . . 1101
6.154.8.7 Just-in-time compilation . . . 1102
6.154.9 Warped VRT . . . 1104
6.154.10 Pansharpened VRT . . . 1105
6.154.11 Multidimensional VRT . . . 1108
6.154.11.1 Multidimensional VRT . . . 1108
6.154.12 Processed dataset VRT . . . 1111
6.154.12.1 VRT processed dataset . . . 1111
6.154.13 vrt:// connection string . . . 1117
6.154.14 Multi-threading optimizations . . . 1120
6.154.15 Performance considerations . . . 1120
6.154.16 Driver capabilities . . . 1120
6.155 WCS -- OGC Web Coverage Service . . . 1121
6.155.1 Driver capabilities . . . 1121
6.155.2 Service description file . . . 1122
6.155.2.1 Range and dimension subsetting . . . 1123
6.155.2.2 Other WCS parameters . . . 1124
6.155.2.3 Open options . . . 1124
6.155.2.4 The cache . . . 1124
6.155.2.5 The WCS: dataset name syntax . . . 1125
6.155.2.6 Time . . . 1125
6.155.2.7 Examples . . . 1125
6.155.3 See Also . . . 1126
6.156 WEBP - WEBP . . . 1127
6.156.1 Driver capabilities . . . 1127
6.156.2 Creation options . . . 1127

6.156.3 See Also . . . 1128
6.157 WMS -- Web Map Services . . . 1128
6.157.1 Driver capabilities . . . 1128
6.157.2 XML description file . . . 1128
6.157.3 Minidrivers . . . 1130
6.157.3.1 WMS . . . 1130
6.157.3.2 TileService . . . 1131
6.157.3.3 WorldWind . . . 1131
6.157.3.4 TMS . . . 1131
6.157.3.5 OnEarth Tiled WMS . . . 1132
6.157.3.6 VirtualEarth . . . 1132
6.157.3.7 ArcGIS REST API . . . 1133
6.157.3.8 Internet Imaging Protocol (IIP) . . . 1133
6.157.3.9 International Image Interoperability Framework Image API (IIIF) . . . 1133
6.157.4 Caching . . . 1133
6.157.5 Configuration options . . . 1134
6.157.6 Examples . . . 1134
6.157.7 Open syntax . . . 1135
6.157.8 Generation of WMS service description XML file . . . 1136
6.157.9 See Also . . . 1136
6.158 WMTS -- OGC Web Map Tile Service . . . 1136
6.158.1 Driver capabilities . . . 1136
6.158.2 Open syntax . . . 1137
6.158.3 Open options . . . 1138
6.158.4 Local service description XML file . . . 1138
6.158.5 GetFeatureInfo request . . . 1141
6.158.6 Generation of WMTS service description XML file . . . 1141
6.158.7 Caching . . . 1142
6.158.8 Configuration options . . . 1142
6.158.9 See Also . . . 1143
6.159 XYZ -- ASCII Gridded XYZ . . . 1143
6.159.1 Open options . . . 1144
6.159.2 Creation options . . . 1144
6.159.3 Driver capabilities . . . 1144
6.159.4 See also . . . 1144
6.160 Zarr . . . 1144
6.160.1 Driver capabilities . . . 1145
6.160.2 Concepts . . . 1145
6.160.3 Dataset name . . . 1145
6.160.4 Zarr stores without directory listing . . . 1146
6.160.5 Sharding support . . . 1146
6.160.6 Multiscales (overviews / pyramids) . . . 1146
6.160.7 Kerchunk reference stores . . . 1146
6.160.8 Compression methods . . . 1147
6.160.9 XArray _ARRAY_DIMENSIONS . . . 1148
6.160.10 NCZarr extensions . . . 1148
6.160.11 Georeferencing encoding (CRS and geotransformation matrix) . . . 1148
6.160.11.1 GDAL convention . . . 1148
6.160.11.2 SPATIAL_PROJ convention . . . 1151
6.160.11.3 netCDF CF conventions . . . 1152
6.160.12 Particularities of the classic raster API . . . 1152
6.160.13 Open options . . . 1152
6.160.14 Multi-threaded caching . . . 1152
6.160.15 Creation options . . . 1153

6.160.16 Add a georeferencing convention to an existing ZARR dataset . . . 1154
6.160.17 Examples . . . 1154
6.160.18 See Also: . . . 1155
6.161 ZMap -- ZMap Plus Grid . . . 1155
6.161.1 Driver capabilities . . . 1155

7 Vector drivers **1157**
7.1 ADBC -- Arrow Database Connectivity . . . 1159
7.1.1 Google BigQuery support . . . 1159
7.1.2 Connection string . . . 1159
7.1.3 Dataset open options . . . 1160
7.1.4 "table_list" special layer . . . 1160
7.1.5 Custom driver entry point . . . 1160
7.1.6 Filtering . . . 1160
7.1.7 Examples . . . 1160
7.1.7.1 Example 1: Convert a Parquet file to GeoPackage . . . 1160
7.1.7.2 Example 2: Convert a DuckDB database to GeoPackage . . . 1161
7.1.8 See Also . . . 1161
7.2 ADBC BigQuery -- Google BigQuery through Arrow Database Connectivity . . . 1161
7.2.1 Runtime requirements . . . 1161
7.2.2 Dataset connection string . . . 1161
7.2.3 Dataset open options . . . 1161
7.2.4 Write support . . . 1162
7.2.5 Layer creation options . . . 1162
7.2.6 Examples . . . 1162
7.2.6.1 Example 1: Listing available tables from a BigQuery dataset . . . 1162
7.2.6.2 Example 2: Displaying the results of a SQL statement . . . 1163
7.2.6.3 Example 3: Converting a subset of a BigQuery table to a GeoPackage file . . . 1163
7.2.6.4 Example 4: Converting a GeoPackage file to a BigQuery table (provided the input table(s) use geographic coordinates): . . . 1163
7.2.6.5 Example 5: Converting a shapefile file to a BigQuery table, doing prior reprojection to WGS 84 (EPSG:4326), and renaming the layer . . . 1163
7.2.7 See Also . . . 1163
7.3 Artificial intelligence powered vector driver . . . 1163
7.3.1 Examples . . . 1164
7.4 AmigoCloud . . . 1164
7.4.1 Driver capabilities . . . 1164
7.4.2 Dataset name syntax . . . 1165
7.4.3 Authentication . . . 1165
7.4.4 Geometry . . . 1165
7.4.5 Filtering . . . 1165
7.4.6 Write support . . . 1165
7.4.7 Layer creation options . . . 1166
7.4.8 Configuration options . . . 1166
7.4.9 Examples . . . 1166
7.4.9.1 Example 1: Different ways to provide AmigoCloud API token . . . 1166
7.4.9.2 Example 2: Show list of datasets . . . 1166
7.4.9.3 Example 3: Accessing data from a list of datasets . . . 1167
7.4.9.4 Example 4: Creating and populating a table from a shapefile . . . 1167
7.4.9.5 Example 5: Append the data to an existing table (dataset id: 12345) from a shapefile 1167
7.4.9.6 Example 6: Overwriting the data of an existing table (dataset id: 12345) with data from a shapefile . . . 1167
7.4.9.7 Example 7: Delete existing dataset (dataset id: 12345) and create a new one with data from a shapefile . . . 1167

7.4.9.8 Example 8: Overwriting the data of an existing table (dataset id: 12345) with data from a shapefile. . . . 1167
7.4.10 See Also . . . 1167
7.5 (Geo)Arrow IPC File Format / Stream . . . 1167
7.5.1 Driver capabilities . . . 1168
7.5.2 Open options . . . 1168
7.5.3 Creation issues . . . 1169
7.5.4 Layer creation options . . . 1169
7.5.5 Conda-forge package . . . 1169
7.5.6 Standalone plugin compilation . . . 1169
7.5.7 Arrow VSI file system . . . 1170
7.5.8 Links . . . 1170
7.6 Arc/Info Binary Coverage . . . 1170
7.6.1 Driver capabilities . . . 1171
7.6.2 See Also . . . 1171
7.7 Arc/Info E00 (ASCII) Coverage . . . 1171
7.7.1 Driver capabilities . . . 1172
7.7.2 See Also . . . 1172
7.8 CAD -- AutoCAD DWG . . . 1172
7.8.1 Driver capabilities . . . 1173
7.8.2 Supported Elements . . . 1173
7.8.3 See Also . . . 1173
7.9 Carto . . . 1174
7.9.1 Driver capabilities . . . 1174
7.9.2 Dataset name syntax . . . 1174
7.9.3 Authentication . . . 1174
7.9.4 Geometry . . . 1175
7.9.5 Filtering . . . 1175
7.9.6 Paging . . . 1175
7.9.7 Write support . . . 1175
7.9.8 SQL . . . 1175
7.9.9 Open options . . . 1176
7.9.10 Layer creation options . . . 1176
7.9.11 Configuration options . . . 1176
7.9.12 Examples . . . 1176
7.9.12.1 Example 1: Accessing data from a public table . . . 1176
7.9.12.2 Example 2: Creating and populating a table from a shapefile . . . 1177
7.9.12.3 Example 3: Creating and populating a table from a CSV containing geometries on EPSG:4326 . . . 1177
7.9.13 See Also . . . 1177
7.10 Comma Separated Value (.csv) . . . 1177
7.10.1 Driver capabilities . . . 1178
7.10.2 Format . . . 1178
7.10.3 Reading CSV containing spatial information . . . 1179
7.10.3.1 Building point geometries . . . 1179
7.10.3.2 Building line geometries . . . 1180
7.10.4 VSI Virtual File System API support . . . 1181
7.10.5 Open options . . . 1181
7.10.6 Creation Issues . . . 1182
7.10.7 Layer Creation options . . . 1182
7.10.8 Configuration options . . . 1183
7.10.8.1 Examples . . . 1183
7.10.9 Particular datasources . . . 1184
7.10.10 Other Notes . . . 1184

7.11 CSW - OGC CSW (Catalog Service for the Web) . . . 1184
7.11.1 Driver capabilities . . . 1184
7.11.2 Dataset name syntax . . . 1184
7.11.3 Filtering . . . 1184
7.11.4 Issues . . . 1185
7.11.5 Open options . . . 1185
7.11.6 Configuration options . . . 1185
7.11.7 Examples . . . 1185
7.11.7.1 Example 1: Listing all the records of a CSW server . . . 1185
7.11.7.2 Example 2: Listing records of a CSW server matching a spatial and attribute filter: . . 1185
7.11.7.3 Example 3: Listing all the records of a CSW server that match a text on any text field 1185
7.11.7.4 Example 4: Listing all the records of a CSW server as ISO 19115/19119 . . . 1186
7.11.8 See Also . . . 1186
7.12 Microstation DGN . . . 1186
7.12.1 Driver capabilities . . . 1186
7.12.2 Supported Elements . . . 1187
7.12.3 MSLINK . . . 1187
7.12.4 User data linkage . . . 1187
7.12.5 Styling Information . . . 1187
7.12.6 Creation Issues . . . 1188
7.12.7 Open options . . . 1188
7.12.8 Dataset creation options . . . 1188
7.13 Microstation DGN v8 . . . 1189
7.13.1 Driver capabilities . . . 1190
7.13.2 Supported Elements . . . 1190
7.13.3 User data linkage . . . 1190
7.13.4 Styling Information . . . 1191
7.13.5 Metadata . . . 1191
7.13.6 Creation Issues . . . 1191
7.13.7 Dataset creation options . . . 1191
7.13.8 Layer creation options . . . 1192
7.13.9 Building . . . 1192
7.13.9.1 Open Design Alliance - ODA Platform . . . 1192
7.14 AutoCAD DWG . . . 1196
7.14.1 Configuration options . . . 1196
7.14.2 Building . . . 1197
7.15 AutoCAD DXF . . . 1197
7.15.1 Driver capabilities . . . 1197
7.15.2 DXF Reader . . . 1197
7.15.2.1 Supported Entities . . . 1198
7.15.2.2 DXF_INLINE_BLOCKS . . . 1199
7.15.2.3 3D Extensibility . . . 1200
7.15.2.4 Character Encodings . . . 1200
7.15.3 Open options . . . 1200
7.15.4 DXF Writer . . . 1201
7.15.4.1 Block References . . . 1202
7.15.4.2 Layer Definitions . . . 1202
7.15.4.3 Linetype Definitions . . . 1202
7.15.5 Binary DXF to ASCII DXF direct translation . . . 1203
7.15.6 See also . . . 1203
7.16 Google Earth Engine Data API . . . 1203
7.16.1 Driver capabilities . . . 1203
7.16.2 Dataset name syntax . . . 1203
7.16.3 Open options . . . 1203

7.16.4 Authentication methods . . . 1204
7.16.5 Configuration options . . . 1204
7.16.6 Attributes . . . 1205
7.16.6.1 Geometry . . . 1206
7.16.6.2 Filtering . . . 1206
7.16.6.3 Paging . . . 1206
7.16.6.4 Extent and feature count . . . 1206
7.16.6.5 Examples . . . 1206
7.16.7 See Also: . . . 1206
7.17 EDIGEO . . . 1207
7.17.1 Driver capabilities . . . 1207
7.17.2 Configuration options . . . 1207
7.17.3 Labels . . . 1207
7.17.4 See Also . . . 1208
7.18 Elasticsearch: Geographically Encoded Objects for Elasticsearch . . . 1208
7.18.1 Driver capabilities . . . 1208
7.18.2 Opening dataset name syntax . . . 1209
7.18.3 Layer open options . . . 1209
7.18.4 Elasticsearch vs OGR concepts . . . 1210
7.18.5 Field definitions . . . 1210
7.18.6 Geometry types . . . 1210
7.18.7 Filtering . . . 1210
7.18.8 Paging . . . 1210
7.18.9 Schema . . . 1210
7.18.10 Feature ID . . . 1211
7.18.11 ExecuteSQL() interface . . . 1211
7.18.12 Aggregations . . . 1212
7.18.13 Multi-target layers . . . 1213
7.18.14 Getting metadata . . . 1213
7.18.15 Write support . . . 1213
7.18.16 Spatial reference system . . . 1213
7.18.17 Layer creation options . . . 1214
7.18.18 Configuration options . . . 1215
7.18.19 Examples . . . 1215
7.18.19.1 Example 1: Open the local store . . . 1215
7.18.19.2 Example 2: Open a remote store . . . 1215
7.18.19.3 Example 3: Filtering on a Elastic Search field . . . 1215
7.18.19.4 Example 4: Basic aggregation . . . 1216
7.18.19.5 Example 5: Load an Elasticsearch index with a shapefile . . . 1216
7.18.19.6 Example 6: Create a Mapping File . . . 1216
7.18.19.7 Example 7: Read the Mapping File . . . 1216
7.18.19.8 Example 8: Bulk Uploading (for larger datasets) . . . 1216
7.18.19.9 Example 9: Overwrite the current Index . . . 1216
7.18.20 See Also . . . 1216
7.19 ESRIJSON / FeatureService driver . . . 1217
7.19.1 Driver capabilities . . . 1217
7.19.2 Datasource . . . 1217
7.19.3 Open options . . . 1218
7.19.4 Examples . . . 1218
7.19.4.1 Example 1: Read the result of a FeatureService request against a GeoServices REST server . . . 1218
7.19.5 See Also . . . 1218
7.20 ESRI File Geodatabase (FileGDB) . . . 1218
7.20.1 Driver capabilities . . . 1218

7.20.2 Requirements . . . 1219
7.20.3 SQL support . . . 1219
7.20.3.1 Special SQL requests . . . 1219
7.20.4 Field domains . . . 1219
7.20.5 Relationships . . . 1219
7.20.6 Hierarchical organization . . . 1219
7.20.7 Geometry coordinate precision . . . 1219
7.20.8 Limitations . . . 1220
7.20.9 Links . . . 1220
7.21 FlatGeobuf . . . 1220
7.21.1 Driver capabilities . . . 1220
7.21.2 Multi layer support . . . 1221
7.21.3 Open options . . . 1221
7.21.4 Dataset Creation Options . . . 1221
7.21.5 Layer Creation Options . . . 1221
7.21.6 Creation Issues . . . 1221
7.21.7 Examples . . . 1222
7.21.7.1 Example 1: Simple translation of a shapefile into a FlatGeobuf file . . . 1222
7.21.7.2 Example 2: Conversion of a Geopackage file with multiple layers . . . 1222
7.21.8 See Also . . . 1222
7.22 GDALG: GDAL Streamed Algorithm . . . 1222
7.22.1 Driver capabilities . . . 1223
7.23 GeoJSON . . . 1223
7.23.1 Driver capabilities . . . 1224
7.23.2 Datasource . . . 1224
7.23.3 Layer . . . 1224
7.23.4 Feature . . . 1225
7.23.5 Geometry . . . 1225
7.23.6 Configuration options . . . 1225
7.23.7 Open options . . . 1226
7.23.8 Layer creation options . . . 1227
7.23.9 VSI Virtual File System API support . . . 1228
7.23.10 Round-tripping of extra JSON members . . . 1228
7.23.11 RFC 7946 write support . . . 1229
7.23.12 Geometry coordinate precision . . . 1229
7.23.13 Examples . . . 1229
7.23.13.1 Example 1: Dumping content of a .geojson file . . . 1229
7.23.13.2 Example 2: Querying features from remote service with filtering by attribute . . . 1229
7.23.13.3 Example 3: Translating number of features queried from FeatureServer to ESRI Shapefile . . . 1230
7.23.13.4 Example 4: Converting a Shapefile into a RFC 7946 GeoJSON file . . . 1230
7.23.14 See Also . . . 1230
7.24 GeoJSONSeq: sequence of GeoJSON features . . . 1230
7.24.1 Driver capabilities . . . 1230
7.24.2 Datasource . . . 1231
7.24.3 Configuration options . . . 1231
7.24.4 Layer creation options . . . 1231
7.24.5 Geometry coordinate precision . . . 1232
7.24.6 See Also . . . 1232
7.25 GeoRSS : Geographically Encoded Objects for RSS feeds . . . 1232
7.25.1 Driver capabilities . . . 1233
7.25.2 Encoding issues . . . 1233
7.25.3 Field definitions . . . 1234
7.25.4 Creation Issues . . . 1234

7.25.5 Dataset creation options . . . 1235
7.25.6 VSI Virtual File System API support . . . 1235
7.25.7 Examples . . . 1236
7.25.7.1 Example 1: Dump the contents of a GeoRSS datafile . . . 1236
7.25.7.2 Example 2: Translating an Atom document into an RSS document . . . 1236
7.25.7.3 Example 3: Reading the contents of a online GeoRSS feed using Python . . . 1236
7.25.8 See Also . . . 1236
7.26 GML - Geography Markup Language . . . 1237
7.26.1 Driver capabilities . . . 1237
7.26.2 Configuration options . . . 1237
7.26.3 Parsers . . . 1238
7.26.4 CRS support . . . 1238
7.26.5 Schema . . . 1239
7.26.6 Particular GML application schemas . . . 1239
7.26.7 Geometry reading . . . 1240
7.26.8 gml:xlink resolving . . . 1240
7.26.9 TopoSurface interpretation rules [polygons and internal holes] . . . 1241
7.26.10 Encoding issues . . . 1242
7.26.11 Feature id (fid / gml:id) . . . 1242
7.26.12 Performance issues with large multi-layer GML files. . . . 1242
7.26.13 Open options . . . 1243
7.26.14 Creation Issues . . . 1244
7.26.15 Dataset creation options . . . 1244
7.26.16 VSI Virtual File System API support . . . 1246
7.26.17 Syntax of .gfs files . . . 1246
7.26.18 Advanced .gfs syntax . . . 1248
7.26.18.1 Specifying ElementPath to find objects embedded into top level objects . . . 1248
7.26.18.2 Getting XML attributes as OGR fields . . . 1249
7.26.18.3 Using conditions on XML attributes . . . 1250
7.26.19 Registry for GML application schemas . . . 1251
7.26.20 Building junction tables . . . 1252
7.26.21 Reading datasets resulting from a WFS 2.0 join queries . . . 1253
7.26.22 Geometry coordinate precision . . . 1255
7.26.23 Examples . . . 1255
7.26.23.1 Example 1: Writing the results of an Oracle query to GML . . . 1255
7.26.23.2 Example 2: Writing the results of a PostGIS query to GML . . . 1255
7.26.24 See Also . . . 1255
7.26.25 Credits . . . 1255
7.27 GMLAS - Geography Markup Language (GML) driven by application schemas . . . 1256
7.27.1 Driver capabilities . . . 1256
7.27.2 Opening syntax . . . 1256
7.27.3 Mapping of XML structure to OGR layers and fields . . . 1257
7.27.4 Metadata layers . . . 1257
7.27.5 Configuration file . . . 1257
7.27.6 Geometry support . . . 1258
7.27.7 Performance issues with large multi-layer GML files. . . . 1258
7.27.8 Open options . . . 1258
7.27.9 Creation support . . . 1259
7.27.9.1 ogr2ogr behavior . . . 1260
7.27.9.2 Dataset creation options . . . 1260
7.27.10 Examples . . . 1261
7.27.10.1 Example 1: Listing content of a data file . . . 1261
7.27.10.2 Example 2: Converting to PostGIS . . . 1261
7.27.10.3 Example 3: Converting to Spatialite and back to GML . . . 1261

7.27.11 See Also . . . 1261
7.27.12 Credits . . . 1261
7.27.12.1 GMLAS - Mapping examples . . . 1261
7.27.12.2 GMLAS - Metadata layers . . . 1264
7.28 GMT ASCII Vectors (.gmt) . . . 1266
7.28.1 Driver capabilities . . . 1267
7.28.2 Creation Issues . . . 1267
7.29 GPKG -- GeoPackage vector . . . 1267
7.29.1 Driver capabilities . . . 1268
7.29.2 Specification version . . . 1268
7.29.3 Limitations . . . 1268
7.29.4 SQL . . . 1268
7.29.4.1 SQL functions . . . 1269
7.29.4.2 Link with Spatialite . . . 1270
7.29.5 Transaction support . . . 1270
7.29.6 Relationships . . . 1271
7.29.7 Dataset open options . . . 1271
7.29.8 Creation issues . . . 1271
7.29.8.1 Dataset creation options . . . 1272
7.29.8.2 Layer creation options . . . 1272
7.29.9 Configuration options . . . 1273
7.29.10 Metadata . . . 1274
7.29.11 Non-spatial tables . . . 1274
7.29.12 Spatial views . . . 1275
7.29.13 Coordinate Reference Systems . . . 1275
7.29.14 Level of support of GeoPackage Extensions . . . 1276
7.29.15 Compressed files . . . 1276
7.29.16 Geometry coordinate precision . . . 1277
7.29.17 Performance hints . . . 1277
7.29.18 Examples . . . 1277
7.29.18.1 Example 1: Simple translation of a single shapefile into GeoPackage . . . 1277
7.29.18.2 Example 2: Updating an existing GeoPackage file . . . 1278
7.29.18.3 Example 3: Converting a directory of shapefiles into a GeoPackage . . . 1278
7.29.18.4 Example 4: Converting a PostGIS database into a GeoPackage . . . 1278
7.29.18.5 Example 5: Performing a join between two GeoPackage databases . . . 1278
7.29.19 Secure deletion . . . 1278
7.29.20 See Also . . . 1278
7.29.20.1 GeoPackage aspatial extension . . . 1278
7.29.20.2 GeoPackage Spatialite computed geometry column extension . . . 1280
7.30 GPSBabel . . . 1282
7.30.1 Driver capabilities . . . 1282
7.30.2 Read support . . . 1282
7.30.3 Write support . . . 1283
7.30.3.1 Examples . . . 1283
7.30.3.2 See Also . . . 1284
7.31 GPX - GPS Exchange Format . . . 1284
7.31.1 Driver capabilities . . . 1284
7.31.2 Configuration options . . . 1285
7.31.3 Open options . . . 1285
7.31.4 Encoding issues . . . 1286
7.31.5 Extensions element reading . . . 1286
7.31.6 Creation Issues . . . 1287
7.31.7 Layer creation options . . . 1287
7.31.8 Dataset creation options . . . 1287

7.31.9 Issues when translating to Shapefile . . . 1288
7.31.10 VSI Virtual File System API support . . . 1289
7.31.11 Examples . . . 1289
7.31.11.1 Example 1: Dumping the contents of a GPX datafile . . . 1289
7.31.11.2 Example 2: GPX to GPX translation . . . 1289
7.31.11.3 Example 3: Use of the <extensions> tag for output . . . 1289
7.31.11.4 Example 4: Use of ogr2ogr -sql option to remap field names to the ones allowed by the GPX schema . . . 1290
7.31.12 FAQ . . . 1290
7.31.13 See Also . . . 1290
7.32 GRASS Vector Format . . . 1290
7.32.1 See Also . . . 1290
7.33 GTFS - General Transit Feed Specification . . . 1290
7.33.1 Open syntax . . . 1291
7.33.2 Driver capabilities . . . 1291
7.33.3 Examples . . . 1291
7.33.4 Links . . . 1292
7.34 SAP HANA . . . 1292
7.34.1 Driver capabilities . . . 1292
7.34.2 Connecting to a database . . . 1293
7.34.3 SQL statements . . . 1293
7.34.4 Creation Issues . . . 1293
7.34.4.1 Dataset Open options . . . 1293
7.34.4.2 Dataset Creation Options . . . 1294
7.34.4.3 Layer Creation Options . . . 1294
7.34.4.4 Multitenant Database Containers . . . 1295
7.34.5 Examples . . . 1295
7.34.5.1 Example 1: List HANA layers on a specified host using ogrinfo . . . 1295
7.34.5.2 Example 2: Print summary information about a the planet_osm_line layer using ogrinfo 1295
7.34.5.3 Example 3: Export data from the points table to a shapefile called points_output.shp 1296
7.34.5.4 Example 4: Create and populate a table with data taken from a shapefile . . . 1296
7.34.6 For developers . . . 1296
7.34.7 See Also . . . 1296
7.35 IDB . . . 1296
7.35.1 Driver capabilities . . . 1297
7.35.2 Environment variables . . . 1297
7.35.3 Configuration options . . . 1297
7.35.4 Examples . . . 1297
7.35.4.1 Example 1 . . . 1297
7.36 Idrisi Vector (.VCT) . . . 1298
7.36.1 Driver capabilities . . . 1298
7.37 "INTERLIS 1" and "INTERLIS 2" drivers . . . 1298
7.37.1 Driver capabilities . . . 1299
7.37.2 Dataset open options . . . 1299
7.37.3 Model support . . . 1300
7.37.3.1 Arc interpolation . . . 1300
7.37.4 Other Notes . . . 1301
7.38 JML: OpenJUMP JML format . . . 1301
7.38.1 Driver capabilities . . . 1301
7.38.2 Encoding issues . . . 1301
7.38.3 Styling . . . 1302
7.38.4 Creation Issues . . . 1302
7.38.5 See Also . . . 1302
7.38.6 Credits . . . 1302

7.39 JSONFG -- OGC Features and Geometries JSON . . . 1302
7.39.1 Driver capabilities . . . 1303
7.39.2 Connection string . . . 1303
7.39.3 CRS support . . . 1303
7.39.4 Time support . . . 1304
7.39.5 Circular-arc geometry support . . . 1304
7.39.6 Geometry measure support . . . 1304
7.39.7 Configuration options . . . 1304
7.39.8 Open options . . . 1304
7.39.9 Dataset creation options . . . 1305
7.39.10 Layer creation options . . . 1305
7.39.11 VSI Virtual File System API support . . . 1305
7.39.12 Geometry coordinate precision . . . 1305
7.39.13 See Also . . . 1306
7.40 KML - Keyhole Markup Language . . . 1306
7.40.1 Driver capabilities . . . 1307
7.40.1.1 KML Reading . . . 1307
7.40.1.2 KML Writing . . . 1307
7.40.1.3 Encoding issues . . . 1307
7.40.1.4 Creation Options . . . 1308
7.40.2 VSI Virtual File System API support . . . 1308
7.40.3 Examples . . . 1308
7.40.3.1 Example 1: Writing the results of a PostGIS query to KML . . . 1308
7.40.3.2 Example 2: Dumping the contents of a .kml file as OGR sees it . . . 1308
7.40.4 Caveats . . . 1308
7.41 LIBKML Driver (.kml .kmz) . . . 1309
7.41.1 Driver capabilities . . . 1309
7.41.2 Datasource . . . 1309
7.41.2.1 StyleTable . . . 1310
7.41.2.2 Dataset creation options . . . 1310
7.41.3 Layer . . . 1311
7.41.3.1 Style . . . 1311
7.41.3.2 Schema . . . 1311
7.41.3.3 Layer creation options . . . 1311
7.41.4 Feature . . . 1313
7.41.4.1 Style . . . 1314
7.41.5 Fields . . . 1314
7.41.6 Geometry . . . 1316
7.41.7 VSI Virtual File System API support . . . 1317
7.41.8 Examples . . . 1317
7.42 Dutch Kadaster LV BAG 2.0 Extract . . . 1319
7.42.1 Open options . . . 1320
7.42.2 VSI Virtual File System API support . . . 1320
7.42.3 Driver capabilities . . . 1320
7.42.4 Examples . . . 1320
7.42.4.1 Example 1: Dumping the results of a LV BAG extract to WGS84 in GeoJSON . . . 1320
7.42.4.2 Example 2: Dumping contents of extract file as OGR sees it . . . 1320
7.42.4.3 Example 3: Inserting repaired features from LV BAG extract archive into PostgreSQL as WGS84 geometries . . . 1320
7.42.4.4 Example 4: Loading a LV BAG extract directory into Postgres . . . 1320
7.42.4.5 Example 5: Creating a GeoPackage from 'Nummeraanduiding' dataset . . . 1321
7.42.5 See Also . . . 1321
7.43 MapML . . . 1321
7.43.1 Driver capabilities . . . 1321

7.43.2 Read support . . . 1322
7.43.3 Write support . . . 1322
7.43.4 Dataset creation options . . . 1322
7.43.5 Links . . . 1322
7.44 MEM -- In Memory datasets . . . 1323
7.44.1 Driver capabilities . . . 1323
7.44.2 Creation Issues . . . 1323
7.44.2.1 Layer creation options . . . 1324
7.45 Memory (deprecated) . . . 1324
7.46 MiraMon Vector . . . 1324
7.46.1 Driver capabilities . . . 1325
7.46.2 Overview of MiraMon format . . . 1325
7.46.3 Encoding . . . 1327
7.46.4 Creation Issues . . . 1327
7.46.5 Connection string . . . 1328
7.46.6 Field sizes . . . 1328
7.46.7 Size Issues . . . 1328
7.46.8 Open options . . . 1328
7.46.9 Dataset creation options . . . 1328
7.46.10 Layer creation options . . . 1328
7.46.11 Examples . . . 1329
7.46.11.1 Example 1 . . . 1329
7.46.11.2 Example 2 . . . 1329
7.46.11.3 Example 3 . . . 1329
7.46.11.4 Example 4 . . . 1329
7.46.11.5 Example 5 . . . 1329
7.46.12 See Also . . . 1329
7.47 MapInfo TAB and MIF/MID . . . 1330
7.47.1 Driver capabilities . . . 1330
7.47.2 Creation Issues . . . 1330
7.47.2.1 Dataset Creation Options . . . 1331
7.47.2.2 Layer Creation Options . . . 1332
7.47.2.3 Configuration options . . . 1333
7.47.2.4 See Also . . . 1333
7.48 MongoDBv3 . . . 1333
7.48.1 Driver capabilities . . . 1333
7.48.2 MongoDB vs OGR concepts . . . 1334
7.48.3 Dataset name syntax . . . 1334
7.48.4 Filtering . . . 1335
7.48.5 Paging . . . 1335
7.48.6 Schema . . . 1335
7.48.7 Feature ID . . . 1335
7.48.8 ExecuteSQL() interface . . . 1335
7.48.9 Write support . . . 1335
7.48.10 Layer creation options . . . 1336
7.48.11 Examples . . . 1336
7.48.11.1 Example 1: Listing the tables of a MongoDB database . . . 1336
7.48.11.2 Example 2: Filtering on a MongoDB field . . . 1336
7.48.11.3 Example 3: Creating and populating a collection from a shapefile . . . 1336
7.48.12 Build instructions . . . 1336
7.48.13 See Also . . . 1337
7.49 MSSQLSpatial - Microsoft SQL Server Spatial Database . . . 1337
7.49.1 Driver capabilities . . . 1337
7.49.2 Connecting to a database . . . 1337

7.49.3 Layers . . . 1338
7.49.4 SQL statements . . . 1338
7.49.5 Creation Issues . . . 1338
7.49.5.1 Layer Creation Options . . . 1338
7.49.6 Configuration options . . . 1339
7.49.7 Transaction support . . . 1340
7.49.8 Examples . . . 1340
7.49.8.1 Example 1: Creating a layer from an OGR data source . . . 1340
7.49.8.2 Example 2: Connecting to a layer and dumping the contents . . . 1340
7.49.8.3 Example 3: Connecting with username/password . . . 1340
7.49.8.4 Example 4: Creating a layer from an OGR data source using the CLI . . . 1340
7.49.8.5 Example 5: Connecting with username/password stored in environment variables . 1341
7.49.8.6 Example 6: Selecting from a database table and writing to a GeoPackage file . . . 1341
7.49.8.7 Example 7: Check if bulk insert (BCP) mode is available . . . 1342
7.50 MVT: Mapbox Vector Tiles . . . 1342
7.50.1 Driver capabilities . . . 1343
7.50.2 Connection strings . . . 1343
7.50.3 metadata.json . . . 1343
7.50.4 Opening options . . . 1345
7.50.5 Creation issues . . . 1345
7.50.6 Dataset creation options . . . 1345
7.50.7 Layer configuration . . . 1347
7.50.8 Layer creation options . . . 1347
7.50.9 Examples . . . 1347
7.50.9.1 Example 1 . . . 1347
7.50.9.2 Example 2 . . . 1347
7.51 MySQL . . . 1348
7.51.1 Driver capabilities . . . 1348
7.51.2 Caveats . . . 1348
7.51.3 Creation Issues . . . 1349
7.51.3.1 Layer Creation Options . . . 1349
7.52 NAS - ALKIS . . . 1351
7.52.1 Driver capabilities . . . 1351
7.53 NetCDF: Network Common Data Form - Vector . . . 1352
7.53.1 Driver capabilities . . . 1352
7.53.2 Conventions and Data Formats . . . 1352
7.53.2.1 Distinguishing the Two Formats . . . 1353
7.53.2.2 CF-1.8 Writing Limitations . . . 1353
7.53.3 Mapping of concepts . . . 1353
7.53.3.1 Field types . . . 1353
7.53.3.2 Layers . . . 1354
7.53.3.3 Strings . . . 1355
7.53.3.4 Geometry . . . 1355
7.53.3.5 "Profile" feature type . . . 1355
7.53.4 Dataset creation options . . . 1356
7.53.5 Layer creation options . . . 1356
7.53.6 XML configuration file . . . 1357
7.53.7 Further Reading . . . 1358
7.53.8 Credits . . . 1359
7.54 NGW -- NextGIS Web . . . 1359
7.54.1 Driver capabilities . . . 1359
7.54.2 Driver . . . 1359
7.54.3 Dataset name syntax . . . 1360
7.54.4 Configuration options . . . 1360

7.54.5 Authentication . . . 1361
7.54.6 Feature . . . 1361
7.54.7 Geometry . . . 1361
7.54.8 Field data types . . . 1361
7.54.9 Paging . . . 1362
7.54.10 Write support . . . 1362
7.54.11 Open options . . . 1362
7.54.12 Dataset creation options . . . 1362
7.54.13 Layer creation options . . . 1363
7.54.14 Metadata . . . 1363
7.54.15 Domains . . . 1363
7.54.16 Filters . . . 1364
7.54.17 Examples . . . 1364
7.54.17.1 Example 1: Reading datasource contents . . . 1364
7.54.17.2 Example 2: Reading layer details . . . 1365
7.54.17.3 Example 3: Creating and populating a vector layer from a shapefile in existing resource group with identifier 1730. . . . 1365
7.54.17.4 Example 4 . . . 1365
7.54.18 See also . . . 1365
7.55 UK .NTF . . . 1365
7.55.1 Driver capabilities . . . 1366
7.55.2 See Also . . . 1366
7.55.3 Implementation Notes . . . 1366
7.55.3.1 Products (and Layers) Supported . . . 1366
7.55.3.2 Product Schemas . . . 1367
7.55.3.3 Special Attributes . . . 1368
7.55.3.4 Generic Products . . . 1368
7.56 OGC API - Features . . . 1369
7.56.1 Driver capabilities . . . 1369
7.56.2 Dataset name syntax . . . 1369
7.56.3 Layer schema . . . 1370
7.56.4 Filtering . . . 1370
7.56.5 CRS support . . . 1370
7.56.6 Open options . . . 1370
7.56.7 Examples . . . 1371
7.56.7.1 Example 1: Listing the types of a OGC API - Features server . . . 1371
7.56.7.2 Example 2: Listing the summary information of a OGC API - Features layer . . . 1371
7.56.7.3 Example 3: Filtering on a property . . . 1372
7.56.7.4 Example 4: Spatial filtering . . . 1373
7.56.8 See Also . . . 1373
7.57 Oracle Spatial . . . 1374
7.57.1 Driver capabilities . . . 1374
7.57.2 SQL Issues . . . 1374
7.57.3 Caveats . . . 1375
7.57.4 Creation Issues . . . 1375
7.57.5 Configuration options . . . 1375
7.57.5.1 Layer Creation Options . . . 1376
7.57.5.2 Layer Open Options . . . 1377
7.57.6 Examples . . . 1377
7.57.6.1 Example 1 . . . 1377
7.57.6.2 Example 2 . . . 1377
7.57.6.3 Example 3 . . . 1377
7.57.6.4 Example 4 . . . 1378
7.57.6.5 Standalone plugin compilation . . . 1378

7.57.6.6 Credits . . . 1378
7.58 ODBC RDBMS . . . 1378
7.58.1 Dataset open options . . . 1379
7.58.2 Driver capabilities . . . 1379
7.58.3 Access Databases (.MDB) support . . . 1379
7.58.4 Creation Issues . . . 1380
7.58.4.1 See Also . . . 1380
7.59 ODS - Open Document Spreadsheet . . . 1380
7.59.1 Driver capabilities . . . 1380
7.59.2 Open options . . . 1380
7.59.3 Configuration options . . . 1381
7.60 ESRI File Geodatabase vector (OpenFileGDB) . . . 1381
7.60.1 Driver capabilities . . . 1381
7.60.2 Spatial filtering . . . 1382
7.60.3 SQL support . . . 1382
7.60.3.1 Special SQL requests . . . 1382
7.60.4 Configuration options . . . 1382
7.60.5 Dataset open options . . . 1383
7.60.6 Dataset Creation Options . . . 1383
7.60.7 Layer Creation Options . . . 1383
7.60.8 64-bit integer field support . . . 1384
7.60.9 Field domains . . . 1384
7.60.10 Relationships . . . 1384
7.60.11 Hierarchical organization . . . 1384
7.60.12 Transaction support . . . 1384
7.60.13 Geometry coordinate precision . . . 1385
7.60.14 Advantages of the OpenFileGDB driver, compared to the FileGDB driver . . . 1385
7.60.15 Limitations . . . 1385
7.60.16 Examples . . . 1385
7.60.16.1 Example 1: Reading a layer from FileGDB and load into PostGIS . . . 1385
7.60.16.2 Example 2: Getting detailed info for FileGDB . . . 1385
7.60.16.3 Example 3: Getting detailed info for a zipped FileGDB . . . 1385
7.60.17 Links . . . 1386
7.60.18 Credits . . . 1386
7.61 OSM - OpenStreetMap XML and PBF . . . 1386
7.61.1 Driver capabilities . . . 1386
7.61.2 Configuration . . . 1387
7.61.2.1 "other_tags" field . . . 1387
7.61.2.2 "all_tags" field . . . 1387
7.61.3 Configuration options . . . 1387
7.61.4 Interleaved reading . . . 1388
7.61.5 Spatial filtering . . . 1388
7.61.6 Reading .osm.bz2 files and/or online files . . . 1388
7.61.7 Open options . . . 1389
7.61.8 Examples . . . 1389
7.61.8.1 Example 1 . . . 1389
7.61.8.2 Example 2 . . . 1389
7.61.8.3 Example 3 . . . 1390
7.61.9 See Also . . . 1390
7.62 (Geo)Parquet . . . 1390
7.62.1 Driver capabilities . . . 1391
7.62.2 Open options . . . 1391
7.62.3 Creation issues . . . 1391
7.62.4 Layer creation options . . . 1391

7.62.5 SQL support . . . 1393
7.62.6 Dataset/partitioning read support . . . 1393
7.62.7 Metadata . . . 1393
7.62.8 Multithreading . . . 1393
7.62.9 Update support . . . 1394
7.62.10 Validation script . . . 1394
7.62.11 Conda-forge package . . . 1394
7.62.11.1 Standalone plugin compilation . . . 1394
7.62.12 Links . . . 1394
7.63 PDF -- Geospatial PDF . . . 1395
7.63.1 Driver capabilities . . . 1395
7.63.2 Vector support . . . 1395
7.63.3 Feature style support . . . 1395
7.63.4 Configuration options . . . 1396
7.63.5 See Also . . . 1396
7.64 PDS - Planetary Data Systems TABLE . . . 1396
7.64.1 Driver capabilities . . . 1397
7.64.2 See Also . . . 1397
7.65 PostgreSQL SQL Dump . . . 1397
7.65.1 Driver capabilities . . . 1398
7.65.2 Creation options . . . 1398
7.65.2.1 Dataset Creation Options . . . 1398
7.65.2.2 Layer Creation Options . . . 1398
7.65.2.3 Environment variables . . . 1400
7.65.2.4 VSI Virtual File System API support . . . 1400
7.65.3 Examples . . . 1400
7.65.3.1 Example 1: Simple translation of a shapefile into PostgreSQL into a file abc.sql . . . 1400
7.65.3.2 Example 2: Piping the output of the PGDump driver into the psql utility. . . . 1400
7.65.3.3 See Also . . . 1400
7.66 ESRI Personal GeoDatabase . . . 1401
7.66.1 Special SQL requests . . . 1401
7.66.2 Configuration options . . . 1401
7.66.3 Dataset open options . . . 1402
7.66.4 Driver capabilities . . . 1402
7.66.5 Field domains . . . 1402
7.66.6 Relationships . . . 1402
7.66.7 How to use PGeo driver with unixODBC and MDB Tools (on Unix and Linux) . . . 1402
7.66.7.1 Prerequisites . . . 1402
7.66.7.2 Configuration . . . 1402
7.66.7.3 Testing PGeo driver with ogrinfo . . . 1404
7.66.8 Resources . . . 1404
7.67 PostgreSQL / PostGIS . . . 1404
7.67.1 Driver capabilities . . . 1405
7.67.2 Connecting to a database . . . 1405
7.67.3 Geometry columns . . . 1406
7.67.4 SQL statements . . . 1406
7.67.5 Creation Issues . . . 1406
7.67.6 Updating existing tables . . . 1406
7.67.6.1 Dataset open options . . . 1407
7.67.6.2 Dataset Creation Options . . . 1407
7.67.6.3 Layer Creation Options . . . 1407
7.67.6.4 Configuration Options . . . 1409
7.67.6.5 Examples . . . 1410
7.67.7 FAQs . . . 1411

7.67.8 See Also . . . 1411
7.67.8.1 PostgreSQL / PostGIS - Advanced Driver Information . . . 1411
7.68 PLScenes (Planet Labs Scenes/Catalog API) . . . 1416
7.68.1 PLScenes (Planet Labs Scenes), Data V1 API . . . 1417
7.68.1.1 Driver capabilities . . . 1417
7.68.1.2 Dataset name syntax . . . 1417
7.68.1.3 Open options . . . 1417
7.68.1.4 Configuration options . . . 1418
7.68.1.5 Attributes . . . 1418
7.68.1.6 Raster access . . . 1419
7.68.1.7 See Also . . . 1420
7.68.2 See Also . . . 1420
7.69 PMTiles . . . 1420
7.69.1 Driver capabilities . . . 1420
7.69.2 Opening options . . . 1421
7.69.3 Creation issues . . . 1421
7.69.4 Dataset creation options . . . 1421
7.69.5 Layer configuration . . . 1422
7.69.6 Layer creation options . . . 1422
7.69.7 /vsipmtiles/ virtual file system . . . 1423
7.69.8 Examples . . . 1424
7.69.8.1 Example 1: Simple translation of a single shapefile into PMTiles . . . 1424
7.69.8.2 Example 2: Merge all PostgreSQL/PostGIS tables in a schema into a single PMTiles file . . . 1424
7.69.9 See Also . . . 1425
7.70 IHO S-57 (ENC) . . . 1425
7.70.1 Driver capabilities . . . 1425
7.70.2 Feature Translation . . . 1426
7.70.3 DSID layer . . . 1427
7.70.4 Soundings . . . 1427
7.70.5 S57 Control Options . . . 1427
7.70.6 S-57 Export . . . 1428
7.70.7 See Also . . . 1429
7.71 IHO S-101 (ENC) . . . 1429
7.71.1 Driver capabilities . . . 1429
7.71.2 Open Options . . . 1430
7.71.3 Layers . . . 1430
7.71.3.1 `informationType` layer . . . 1430
7.71.3.2 `Point2D` layer . . . 1431
7.71.3.3 `Point3D_{vertical_crs_name}` layers . . . 1431
7.71.3.4 `MultiPoint2D` layer . . . 1431
7.71.3.5 `MultiPoint3D_{vertical_crs_name}` layers . . . 1431
7.71.3.6 `Curve` layer . . . 1431
7.71.3.7 `CompositeCurve` layer . . . 1431
7.71.3.8 `Surface` layer . . . 1432
7.71.3.9 Feature type layers . . . 1432
7.71.4 Dataset metadata . . . 1433
7.71.5 Feature Catalog . . . 1433
7.71.6 See Also . . . 1433
7.71.7 Credits . . . 1434
7.72 Selafin files . . . 1434
7.72.1 Driver capabilities . . . 1434
7.72.2 Magic bytes . . . 1434
7.72.3 Format . . . 1435

7.72.3.1 Elements . . . 1435
7.72.3.2 Full structure . . . 1436
7.72.4 Mapping between file and layers . . . 1437
7.72.4.1 Layers in a Selafin datasource . . . 1437
7.72.4.2 Constraints on layers . . . 1438
7.72.5 Layer filtering specification . . . 1438
7.72.6 Datasource creation options . . . 1439
7.72.7 Layer creation options . . . 1439
7.72.8 Notes about the creation and the update of a Selafin datasource . . . 1439
7.72.9 VSI Virtual File System API support . . . 1440
7.72.10 Other notes . . . 1440
7.73 ESRI Shapefile / DBF . . . 1441
7.73.1 Driver capabilities . . . 1441
7.73.2 Encoding . . . 1442
7.73.3 Spatial and attribute indexing . . . 1442
7.73.4 Creation Issues . . . 1443
7.73.5 Field sizes . . . 1444
7.73.6 Spatial extent . . . 1444
7.73.7 Size Issues . . . 1444
7.73.8 Compressed files . . . 1444
7.73.9 Open options . . . 1444
7.73.10 Dataset creation options . . . 1445
7.73.11 Layer creation options . . . 1445
7.73.12 Configuration options . . . 1445
7.73.13 Examples . . . 1446
7.73.13.1 Example 1: Merging two shapefiles 'file1.shp' and 'file2.shp' . . . 1446
7.73.13.2 Example 2: Building a spatial index . . . 1446
7.73.13.3 Example 3: Resizing columns of a DBF file to their optimal size . . . 1446
7.73.14 See Also . . . 1446
7.74 Norwegian SOSI Standard . . . 1446
7.74.1 Open options . . . 1446
7.74.1.1 Examples . . . 1447
7.75 SQLite / Spatialite RDBMS . . . 1447
7.75.1 Driver capabilities . . . 1448
7.75.2 "Regular" SQLite databases . . . 1448
7.75.3 Tables with multiple geometry columns . . . 1449
7.75.4 REGEXP operator . . . 1449
7.75.5 Using the SpatiaLite library (Spatial extension for SQLite) . . . 1449
7.75.6 Opening with 'VirtualShape:' . . . 1449
7.75.7 The SQLite SQL dialect . . . 1449
7.75.8 The VirtualOGR SQLite extension . . . 1449
7.75.9 Creation Issues . . . 1450
7.75.9.1 Transaction support . . . 1450
7.75.10 Relationships . . . 1450
7.75.10.1 Dataset open options . . . 1451
7.75.10.2 Database creation options . . . 1451
7.75.10.3 Layer creation options . . . 1451
7.75.11 Configuration options . . . 1452
7.75.12 Performance hints . . . 1453
7.75.13 Secure deletion . . . 1454
7.75.14 Examples . . . 1454
7.75.14.1 Example 1: Converting a non-spatial SQLite table into a GeoPackage . . . 1454
7.75.14.2 Example 2: Performing a join between two SQLite/Spatialite databases . . . 1454
7.75.15 Credits . . . 1454

7.75.16 Links . . . 1454
7.76 Storage and eXchange Format - SXF . . . 1454
7.76.1 Driver capabilities . . . 1455
7.76.2 See Also . . . 1455
7.77 U.S. Census TIGER/Line . . . 1456
7.77.1 Driver capabilities . . . 1457
7.77.2 Feature Representation . . . 1457
7.77.2.1 CompleteChain . . . 1457
7.77.2.2 AltName . . . 1457
7.77.2.3 FeatureIds . . . 1458
7.77.2.4 ZipCodes . . . 1458
7.77.2.5 Landmarks . . . 1458
7.77.2.6 AreaLandmarks . . . 1458
7.77.2.7 KeyFeatures . . . 1458
7.77.2.8 Polygon . . . 1458
7.77.2.9 EntityNames . . . 1458
7.77.2.10 IDHistory . . . 1459
7.77.2.11 PolyChainLink . . . 1459
7.77.2.12 PIP . . . 1459
7.77.2.13 ZipPlus4 . . . 1459
7.77.3 See Also . . . 1459
7.78 TileDB -- TileDB vector . . . 1459
7.78.1 Driver capabilities . . . 1460
7.78.2 Supported datasets . . . 1460
7.78.3 Dataset connection string . . . 1460
7.78.4 Filtering . . . 1460
7.78.5 Arrow C Stream data interface . . . 1461
7.78.6 Creation issues . . . 1461
7.78.7 Open options . . . 1461
7.78.8 Dataset creation options . . . 1461
7.78.9 Layer creation options . . . 1461
7.78.10 See Also . . . 1462
7.79 TopoJSON driver . . . 1462
7.79.1 Driver capabilities . . . 1462
7.79.2 Datasource . . . 1463
7.79.3 Examples . . . 1463
7.79.3.1 Example 1: Reading a TopoJSON file with multiple layers . . . 1463
7.79.3.2 Example 2: Writing a single layer from a TopoJSON file to GeoJSON file . . . 1463
7.79.4 See Also . . . 1463
7.80 VDV - VDV-451/VDV-452/INTREST Data Format . . . 1463
7.80.1 Driver capabilities . . . 1464
7.80.2 Creation issues . . . 1464
7.80.3 Links . . . 1465
7.81 VFK - Czech Cadastral Exchange Data Format . . . 1465
7.81.1 Driver capabilities . . . 1465
7.81.2 Open options . . . 1465
7.81.3 Configuration options . . . 1466
7.81.3.1 Internal working and performance tweaking . . . 1466
7.81.3.2 Examples . . . 1466
7.81.4 Datasource name . . . 1467
7.81.5 Layer names . . . 1467
7.81.6 Filters . . . 1467
7.81.6.1 Attribute filter . . . 1467
7.81.6.2 Spatial filter . . . 1467

7.81.7 References . . . 1467
7.82 VRT -- OGR Virtual Format . . . 1467
7.82.1 Driver capabilities . . . 1468
7.82.2 Creation Issues . . . 1468
7.82.3 Virtual File Format . . . 1468
7.82.3.1 Metadata element . . . 1468
7.82.3.2 OGRVRTLayer element . . . 1468
7.82.3.3 OGRVRTWarpedLayer element . . . 1471
7.82.3.4 OGRVRTUnionLayer element . . . 1471
7.82.4 Examples . . . 1472
7.82.4.1 Example 1: ODBC Point Layer . . . 1472
7.82.4.2 Example 2: Renaming attributes . . . 1472
7.82.4.3 Example 3: Transparent spatial filtering . . . 1473
7.82.4.4 Example 4: Reprojected layer . . . 1473
7.82.4.5 Example 5: Union layer . . . 1473
7.82.4.6 Example 6: SQLite/Spatialite SQL dialect . . . 1474
7.82.4.7 Example 7: Multiple geometry fields . . . 1474
7.82.5 Other Notes . . . 1475
7.83 WAsP - WAsP .map format . . . 1476
7.83.1 Driver capabilities . . . 1476
7.83.2 Configuration options . . . 1476
7.84 WFS - OGC WFS service . . . 1477
7.84.1 Driver capabilities . . . 1477
7.84.2 Dataset name syntax . . . 1477
7.84.3 Request paging . . . 1478
7.84.3.1 Paging with WFS 2.0 . . . 1478
7.84.3.2 Paging with WFS 1.0 or 1.1 . . . 1478
7.84.3.3 Paging options . . . 1479
7.84.4 Filtering . . . 1479
7.84.5 Layer joins . . . 1479
7.84.6 Write support / WFS-T . . . 1479
7.84.7 Write support and OGR transactions . . . 1480
7.84.8 Special SQL commands . . . 1480
7.84.9 Special layer : WFSLayerMetadata . . . 1480
7.84.10 Special layer : WFSGetCapabilities . . . 1480
7.84.11 Open options . . . 1481
7.84.12 Configuration options . . . 1481
7.84.13 Examples . . . 1481
7.84.13.1 Example 1: Listing the types of a WFS server . . . 1481
7.84.13.2 Example 2: Listing the types of a WFS server whose layer structures are cached in a XML file . . . 1481
7.84.13.3 Example 3: Listing the features of the dvg:nw_dvg2_rbz layer, with a spatial filter . 1481
7.84.13.4 Example 4: Retrieving the features of GN "Köln" and "Viersen" from the dvg:nw_dvg2_krs layer . . . 1482
7.84.13.5 Example 5: Displaying layer metadata . . . 1482
7.84.14 See Also . . . 1482
7.85 XLS - MS Excel format . . . 1482
7.85.1 Configuration options . . . 1482
7.85.2 See Also . . . 1483
7.86 XLSX - MS Office Open XML spreadsheet . . . 1483
7.86.1 Driver capabilities . . . 1483
7.86.2 Open options . . . 1483
7.86.3 Configuration options . . . 1484
7.86.4 Examples . . . 1484

7.86.4.1 Example 1 . . . 1484
7.87 XODR -- OpenDRIVE Road Description Format . . . 1484
7.87.1 Driver capabilities . . . 1485
7.87.2 Specification version . . . 1485
7.87.2.1 Supported OpenDRIVE elements . . . 1485
7.87.3 Spatial reference . . . 1485
7.87.4 Limitations . . . 1485
7.87.5 Open options . . . 1485
7.87.6 Examples . . . 1486
7.87.6.1 Example 1: Translating OpenDRIVE road ReferenceLine elements (<planView>) to Shapefile using ogr2ogr. . . . 1486
7.87.6.2 Example 2: Converting the whole OpenDRIVE dataset with all its different layers into a GeoPackage . . . 1486
7.87.6.3 Example 3: Converting the whole OpenDRIVE dataset with custom parameters EPSILON and DISSOLVE_TIN into a GeoPackage: . . . 1486
7.87.7 Convenient usage through docker image . . . 1486
7.87.8 General building notes . . . 1486
7.87.8.1 Verifying a successful build . . . 1487

8 User oriented documentation **1489**
8.1 GDAL Command-line Cookbook . . . 1489
8.1.1 Introduction . . . 1489
8.1.2 General How do I... . . . 1489
8.1.3 Raster How do I... . . . 1490
8.1.4 Vector How do I... . . . 1490
8.1.5 Raster and Vector How do I... . . . 1490
8.2 GDAL CLI Debug and Logging Output . . . 1491
8.2.1 Debug Output . . . 1491
8.2.1.1 Example 1: Convert vector datasets with debug output . . . 1491
8.2.1.2 Example 2: Filter debug output by driver . . . 1492
8.2.2 Logging and Diagnostic Options . . . 1492
8.2.2.1 Example 3: Get raster information with HTTP request logging enabled . . . 1493
8.2.2.2 Example 4: Get raster info with curl logging and timestamps . . . 1493
8.2.2.3 Example 5: Log debug output and errors to a file . . . 1493
8.3 Error Codes . . . 1493
8.3.1 Overview . . . 1493
8.3.2 General Error Codes . . . 1495
8.3.2.1 ERROR 1 Application Defined Error . . . 1495
8.3.2.2 ERROR 2 Out of Memory Error . . . 1496
8.3.2.3 ERROR 3 File I/O Error . . . 1496
8.3.2.4 ERROR 4 Open Failed . . . 1496
8.3.2.5 ERROR 5 Illegal Argument . . . 1497
8.3.2.6 ERROR 6 Not Supported . . . 1497
8.3.2.7 ERROR 7 Assertion Failed . . . 1498
8.3.2.8 ERROR 8 No Write Access . . . 1498
8.3.2.9 ERROR 9 User Interrupted . . . 1498
8.3.3 Network Related Error Codes . . . 1498
8.3.3.1 ERROR 10 HTTP Response . . . 1498
8.3.3.2 ERROR 11 Bucket Not Found . . . 1498
8.3.3.3 ERROR 12 Object Not Found . . . 1498
8.3.3.4 ERROR 13 Access Denied . . . 1498
8.3.3.5 ERROR 14 Invalid Credentials . . . 1498
8.3.3.6 ERROR 15 Signature Does Not Match . . . 1498
8.3.3.7 ERROR 16 Object Storage Generic Error . . . 1499

8.4 Raster Data Model . . . 1499
8.4.1 Dataset . . . 1499
8.4.2 Coordinate System . . . 1499
8.4.3 Affine GeoTransform . . . 1500
8.4.4 GCPs . . . 1500
8.4.5 Metadata . . . 1500
8.4.5.1 SUBDATASETS Domain . . . 1502
8.4.5.2 IMAGE_STRUCTURE Domain . . . 1502
8.4.5.3 RPC Domain . . . 1504
8.4.5.4 IMAGERY Domain (remote sensing) . . . 1505
8.4.5.5 xml: Domains . . . 1505
8.4.6 Raster Band . . . 1505
8.4.7 Color Table . . . 1507
8.4.8 Raster Attribute Table . . . 1508
8.4.9 Overviews . . . 1508
8.5 GDAL Band Algebra . . . 1509
8.5.1 Description . . . 1509
8.5.2 Examples . . . 1510
8.5.2.1 Example 1: Convert a RGB dataset to a graylevel one. . . . 1510
8.5.2.2 Example 2: Compute normalized difference vegetation index (NDVI) . . . 1511
8.5.2.3 Example 3: Normalizing the values of a band to the [0, 1] range using the minimum and maximum of all bands . . . 1511
8.6 Multidimensional Raster Data Model . . . 1512
8.6.1 Group . . . 1512
8.6.2 Attribute . . . 1513
8.6.3 Multidimensional array . . . 1513
8.6.4 Dimension . . . 1513
8.6.5 Data Type . . . 1514
8.6.6 Objects lifetime . . . 1514
8.6.7 Differences with the GDAL 2D raster data model . . . 1516
8.6.8 Bridges between GDAL 2D classic raster data model and multidimensional data model . . . 1516
8.6.9 Applications . . . 1516
8.7 Vector Data Model . . . 1516
8.7.1 Class Overview . . . 1516
8.7.2 Geometry . . . 1517
8.7.2.1 Compatibility issues with non-linear geometries . . . 1517
8.7.3 Spatial Reference . . . 1518
8.7.4 Feature / Feature Definition . . . 1518
8.7.5 Field Definitions . . . 1518
8.7.5.1 Field Domains . . . 1518
8.7.6 Layer . . . 1519
8.7.7 Dataset . . . 1519
8.7.8 Drivers . . . 1520
8.8 Geometry Validity . . . 1520
8.8.1 Validity checking . . . 1520
8.8.2 Geometry repair . . . 1521
8.8.2.1 Invalid geometry examples . . . 1521
8.9 Geographic Networks Data Model . . . 1529
8.9.1 General concept . . . 1529
8.9.1.1 Network . . . 1529
8.9.1.2 Format . . . 1530
8.9.2 Network formats . . . 1530
8.9.2.1 GNMGenericNetwork . . . 1530
8.9.3 Network analysis . . . 1530

8.10 Multi-threading . . . 1531
8.10.1 GDAL API: re-entrant, but (generally) not thread-safe . . . 1531
8.10.2 Thread-safe GDAL dataset instances for raster read-only use cases . . . 1531
8.10.3 GDAL block cache and multi-threading . . . 1531
8.10.4 RAM fragmentation and multi-threading . . . 1532
8.10.5 GDAL and multi-processing . . . 1532
8.11 OGR SQL dialect and SQLITE SQL dialect . . . 1532
8.11.1 Dialects . . . 1532
8.11.1.1 OGR SQL dialect . . . 1532
8.11.1.2 SQL SQLite dialect . . . 1542
8.11.2 ExecuteSQL() . . . 1549
8.12 GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ... 1549
8.12.1 Introduction . . . 1549
8.12.2 Chaining . . . 1549
8.12.3 Drivers supporting virtual file systems . . . 1550
8.12.4 /vsizip/ (.zip archives) . . . 1550
8.12.4.1 Read capabilities . . . 1550
8.12.4.2 Write capabilities . . . 1551
8.12.4.3 SOZip (Seek-Optimized ZIP) . . . 1551
8.12.5 /vsigzip/ (gzipped file) . . . 1551
8.12.6 /vsitar/ (.tar, .tgz archives) . . . 1552
8.12.7 /vsi7z/ (.7z archives) . . . 1552
8.12.8 /vsirar/ (.rar archives) . . . 1553
8.12.9 Network based file systems . . . 1553
8.12.9.1 /vsicurl/ vs /vsicurl_streaming/ vs HTTP pseudo-driver . . . 1553
8.12.9.2 How to set credentials ? . . . 1554
8.12.9.3 Network/cloud-friendliness and file formats . . . 1555
8.12.9.4 /vsicurl/ (http/https/ftp files: random access) . . . 1556
8.12.9.5 /vsicurl_streaming/ (http/https/ftp files: streaming) . . . 1558
8.12.9.6 /vsis3/ (AWS S3 files) . . . 1559
8.12.9.7 /vsis3/ and AWS Instance Metadata Service (IMDS) . . . 1561
8.12.9.8 Configuring /vsis3/ with Minio . . . 1561
8.12.9.9 /vsis3_streaming/ (AWS S3 files: streaming) . . . 1561
8.12.9.10 /vsigs/ (Google Cloud Storage files) . . . 1562
8.12.9.11 /vsigs_streaming/ (Google Cloud Storage files: streaming) . . . 1563
8.12.9.12 /vsiaz/ (Microsoft Azure Blob files) . . . 1563
8.12.9.13 /vsiaz_streaming/ (Microsoft Azure Blob files: streaming) . . . 1565
8.12.9.14 /vsiadls/ (Microsoft Azure Data Lake Storage Gen2) . . . 1565
8.12.9.15 /vsioss/ (Alibaba Cloud OSS files) . . . 1566
8.12.9.16 /vsioss_streaming/ (Alibaba Cloud OSS files: streaming) . . . 1566
8.12.9.17 /vsiswift/ (OpenStack Swift Object Storage) . . . 1566
8.12.9.18 /vsiswift_streaming/ (OpenStack Swift Object Storage: streaming) . . . 1568
8.12.9.19 /vsihdfs/ (Hadoop File System) . . . 1568
8.12.9.20 /vsiwebhdfs/ (Web Hadoop File System REST API) . . . 1568
8.12.10 /vsistdin/ (standard input streaming) . . . 1569
8.12.11 /vsistdout/ (standard output streaming) . . . 1569
8.12.12 /vsimem/ (in-memory files) . . . 1570
8.12.13 /vsisubfile/ (portions of files) . . . 1570
8.12.14 /vsisparse/ (sparse files) . . . 1570
8.12.15 /vsicached/ (File caching) . . . 1571
8.12.16 /vsicrypt/ (encrypted files) . . . 1572
8.13 Feature Style Specification . . . 1572
8.13.1 1. Overview . . . 1572
8.13.1.1 1.1 Style is a property of Feature object . . . 1572

8.13.1.2 1.2 Feature Styles can be stored at 2 levels . . . 1572
8.13.1.3 1.3 Drawing Tools . . . 1572
8.13.1.4 1.4 Feature attributes can be used by style definitions . . . 1573
8.13.1.5 1.5 Tool parameter units . . . 1573
8.13.2 2. Feature Style String . . . 1573
8.13.2.1 2.1 Examples . . . 1573
8.13.2.2 2.2 Style String Syntax . . . 1574
8.13.2.3 2.3 Pen Tool Parameters . . . 1575
8.13.2.4 2.4 Brush Tool Parameters . . . 1576
8.13.2.5 2.5 Symbol Tool Parameters . . . 1578
8.13.2.6 2.6 Label Tool Parameters . . . 1580
8.13.2.7 2.7 Styles Table Format . . . 1582
8.13.2.8 2.8 Using OGR SQL to transfer the style between the data sources . . . 1583
8.13.3 3. OGR Support Classes . . . 1583
8.14 Coordinate epoch support . . . 1585
8.14.1 Dynamic CRS and coordinate epoch . . . 1585
8.14.2 Support in raster and vector formats . . . 1585
8.14.2.1 FlatGeoBuf . . . 1586
8.14.2.2 GeoPackage vector/raster . . . 1586
8.14.2.3 GeoTIFF . . . 1586
8.14.2.4 JPEG2000 . . . 1587
8.14.2.5 Persistent Auxiliary Metadata (.aux.xml) . . . 1587
8.14.2.6 GDAL VRT . . . 1587
8.14.3 Support in utilities . . . 1587
8.14.4 Support in coordinate transformation . . . 1588
8.15 Configuration options . . . 1588
8.15.1 How to set configuration options? . . . 1588
8.15.2 How to detect if the passed configuration option is known to GDAL . . . 1589
8.15.3 GDAL configuration file . . . 1589
8.15.4 Global configuration options . . . 1590
8.15.4.1 Logging . . . 1590
8.15.4.2 Performance and caching . . . 1590
8.15.4.3 Driver management . . . 1591
8.15.4.4 General options . . . 1592
8.15.4.5 Vector related options . . . 1593
8.15.4.6 Networking options . . . 1594
8.15.4.7 Persistent Auxiliary Metadata (PAM) options . . . 1597
8.15.4.8 PROJ options . . . 1597
8.15.5 List of configuration options and where they are documented . . . 1598
8.16 GDAL Python Utilities . . . 1620
8.16.1 Developers . . . 1620
8.17 Migration guide . . . 1620
8.17.1 From GDAL 3.13 to GDAL 3.14 . . . 1620
8.17.2 From GDAL 3.12 to GDAL 3.13 . . . 1620
8.17.3 From GDAL 3.11 to GDAL 3.12 . . . 1621
8.17.4 From GDAL 3.10 to GDAL 3.11 . . . 1622
8.17.5 From GDAL 3.9 to GDAL 3.10 . . . 1623
8.17.6 From GDAL 3.8 to GDAL 3.9 . . . 1623
8.17.7 From GDAL 3.7 to GDAL 3.8 . . . 1624
8.17.8 From GDAL 3.6 to GDAL 3.7 . . . 1624
8.17.9 From GDAL 3.5 to GDAL 3.6 . . . 1625
8.17.10 From GDAL 3.4 to GDAL 3.5 . . . 1625
8.17.11 From GDAL 3.3 to GDAL 3.4 . . . 1625
8.17.12 From GDAL 3.2 to GDAL 3.3 . . . 1626

8.17.13 From GDAL 3.1 to GDAL 3.2 . . . 1626
8.17.14 From GDAL 3.0 to GDAL 3.1 . . . 1626
8.17.15 From GDAL 2.x to GDAL 3.0 . . . 1627
8.18 Security considerations . . . 1627
8.18.1 Classes of potential vulnerabilities . . . 1627
8.18.2 Causes . . . 1627
8.18.3 Situations at risk . . . 1627
8.18.4 Mitigation . . . 1627
8.18.5 Credential related issues . . . 1628
8.18.6 Known issues in API . . . 1628
8.18.7 Known issues in drivers . . . 1628
8.18.7.1 General issues . . . 1628
8.18.7.2 GDAL MEM driver . . . 1629
8.18.7.3 GDAL PDF driver . . . 1629
8.18.7.4 GDAL VRT driver . . . 1629
8.18.7.5 GDAL GTI driver . . . 1629
8.18.7.6 OGR VRT driver . . . 1629

9 API **1631**

10 Tutorials **1633**
10.1 Raster . . . 1633
10.1.1 Raster API tutorial . . . 1633
10.1.1.1 Opening the File . . . 1633
10.1.1.2 Getting Dataset Information . . . 1634
10.1.1.3 Fetching a Raster Band . . . 1636
10.1.1.4 Reading Raster Data . . . 1637
10.1.1.5 Closing the Dataset . . . 1638
10.1.1.6 Techniques for Creating Files . . . 1639
10.1.1.7 Using CreateCopy() . . . 1640
10.1.1.8 Using Create() . . . 1641
10.1.2 Raster driver implementation tutorial . . . 1643
10.1.2.1 Overall Approach . . . 1643
10.1.2.2 Implementing the Dataset . . . 1643
10.1.2.3 Implementing the RasterBand . . . 1647
10.1.2.4 The Driver . . . 1649
10.1.2.5 Adding Driver to GDAL Tree . . . 1651
10.1.2.6 Adding Georeferencing . . . 1651
10.1.2.7 Overviews . . . 1652
10.1.2.8 File Creation . . . 1653
10.1.2.9 RawDataset/RawRasterBand Helper Classes . . . 1658
10.1.2.10 Metadata, and Other Exotic Extensions . . . 1658
10.1.3 GDAL Warp API tutorial (Reprojection, ...) . . . 1659
10.1.3.1 Overview . . . 1659
10.1.3.2 A Simple Reprojection Case . . . 1659
10.1.3.3 Other Warping Options . . . 1660
10.1.3.4 Creating the Output File . . . 1661
10.1.3.5 Performance Optimization . . . 1663
10.1.3.6 Other Masking Options . . . 1664
10.1.4 GDAL Grid Tutorial . . . 1664
10.1.4.1 Introduction to Gridding . . . 1664
10.1.4.2 Interpolation of the Scattered Data . . . 1664
10.1.4.3 Data Metrics Computation . . . 1665
10.1.4.4 Search Ellipse . . . 1666

10.1.5 Geotransform Tutorial . . . 1667
10.1.5.1 Introduction to Geotransforms: . . . 1667
10.1.5.2 Transformation from image coordinate space to georeferenced coordinate space: . . . 1667
10.1.5.3 In case of north up images: . . . 1667
10.1.6 Working with a Digital Terrain Model (DTM) . . . 1668
10.1.6.1 Overview . . . 1668
10.1.6.2 Getting the Data . . . 1668
10.1.6.3 Creating a Raster Tile Index . . . 1669
10.1.6.4 Clipping the DTM . . . 1669
10.1.6.5 Processing the DTM . . . 1671
10.2 Multidimensional raster . . . 1675
10.2.1 Multidimensional raster API tutorial . . . 1675
10.2.1.1 Read the content of an array . . . 1675
10.2.1.2 Other examples . . . 1678
10.3 Vector . . . 1678
10.3.1 Vector API tutorial . . . 1678
10.3.1.1 Reading From OGR . . . 1678
10.3.1.2 Reading From OGR using the Arrow C Stream data interface . . . 1688
10.3.1.3 Writing To OGR . . . 1693
10.3.1.4 Writing to OGR using the Arrow C Data interface . . . 1704
10.3.2 Vector driver implementation tutorial . . . 1707
10.3.2.1 Overall Approach . . . 1707
10.3.2.2 Implementing GDALDriver . . . 1707
10.3.2.3 Basic Read Only Data Source . . . 1708
10.3.2.4 Read Only Layer . . . 1710
10.3.3 Downloading GeoParquet Data from Overture Maps . . . 1713
10.3.3.1 Overview . . . 1713
10.3.3.2 Setting Up a Conda Environment with Parquet Support . . . 1713
10.3.3.3 Listing Overture Data . . . 1715
10.3.3.4 Downloading Data . . . 1717
10.3.4 Working with DuckDB . . . 1720
10.3.4.1 Overview . . . 1720
10.3.4.2 Setup and Configuration . . . 1720
10.3.4.3 Downloading Postal Addresses . . . 1722
10.3.4.4 Creating Hex Bins of Point Data . . . 1724
10.3.5 Vector driver in Python implementation tutorial . . . 1728
10.3.5.1 Introduction . . . 1728
10.3.5.2 Linking mechanism to a Python interpreter . . . 1728
10.3.5.3 Driver location . . . 1728
10.3.5.4 Import section . . . 1728
10.3.5.5 Metadata section . . . 1728
10.3.5.6 Driver class . . . 1729
10.3.5.7 Dataset class . . . 1730
10.3.5.8 Layer class . . . 1731
10.3.5.9 Full example . . . 1734
10.3.5.10 Other examples . . . 1737
10.4 Geographic Network Model . . . 1737
10.4.1 GNM API tutorial . . . 1737
10.4.1.1 Managing networks . . . 1737
10.4.1.2 Analysing networks . . . 1742
10.5 Projections / CRS . . . 1744
10.5.1 Reprojecting Data with GDAL . . . 1744
10.5.1.1 Introduction . . . 1744
10.5.1.2 Raster . . . 1748

10.5.1.3 Vector . . . 1749
10.5.1.4 Further Reading . . . 1749
10.5.2 OGR Coordinate Reference Systems and Coordinate Transformation tutorial . . . 1750
10.5.2.1 OGC WKT Coordinate System Issues . . . 1750
10.5.2.2 Introduction . . . 1755
10.5.2.3 References and applicable standards . . . 1755
10.5.2.4 Defining a Geographic Coordinate Reference System . . . 1755
10.5.2.5 CRS and axis order . . . 1757
10.5.2.6 Defining a Projected CRS . . . 1758
10.5.2.7 Querying Coordinate Reference System . . . 1759
10.5.2.8 Coordinate Transformation . . . 1760
10.5.2.9 Advanced Coordinate Transformation . . . 1761
10.5.2.10 Alternate Interfaces . . . 1762
10.5.2.11 History and implementation considerations . . . 1764
10.6 Workshops . . . 1764
10.7 External tutorials . . . 1764
10.7.1 A Gentle Introduction to GDAL, by Robert Simmon . . . 1764

11 Community **1767**
11.1 Code of Conduct . . . 1767
11.1.1 Code of Conduct . . . 1767
11.1.1.1 Introduction . . . 1767
11.1.1.2 Diversity Statement . . . 1767
11.1.1.3 Anti-Harassment . . . 1768
11.1.1.4 Reporting Guidelines . . . 1769
11.1.1.5 Credits . . . 1769
11.2 AI Tool Use Policy . . . 1769
11.2.1 LLM tool policy . . . 1769
11.2.1.1 Rationale . . . 1769
11.2.1.2 Policy . . . 1769
11.2.1.3 Violations . . . 1770
11.2.1.4 Mitigation . . . 1770
11.3 Mailing List . . . 1770
11.4 GitHub . . . 1770
11.5 Chat . . . 1771
11.6 Social media . . . 1771
11.7 Conferences . . . 1771
11.7.1 Upcoming . . . 1771
11.7.2 Previous . . . 1771
11.8 Governance and Community Participation . . . 1772
11.8.1 OSGeo Project Membership . . . 1772
11.8.2 Project Steering Committee . . . 1772
11.8.3 User survey . . . 1773
11.8.3.1 2024 GDAL user survey . . . 1773
11.8.3.2 2025 GDAL user survey . . . 1773
11.9 GDAL Service Providers . . . 1773
11.9.1 GDAL Service Providers . . . 1773
11.9.1.1 Core Contributors . . . 1773
11.9.1.2 Contributors . . . 1774
11.9.1.3 Other Service Providers . . . 1774
11.9.1.4 How to Add a Company . . . 1775

12 Sponsors **1777**
12.1 Gold . . . 1777

12.2 Silver . . . 1777
12.3 Bronze . . . 1778
12.4 Supporter . . . 1778
12.5 Sponsorship Program . . . 1778
12.5.1 Donations . . . 1779
12.5.1.1 Levels . . . 1779
12.5.1.2 Corporate . . . 1779
12.5.1.3 Individual . . . 1779
12.5.1.4 Government . . . 1779
12.5.1.5 T-shirts . . . 1779
12.5.1.6 Past Sponsors . . . 1780
12.5.2 Administration . . . 1780
12.5.3 Impact . . . 1780
12.5.4 Related Resources . . . 1780
12.5.4.1 Sponsoring FAQ . . . 1781

13 How to contribute? 1783

14 Thanks 1785

Bibliography 1789

Index 1791

CHAPTER

ONE

WHAT IS GDAL?

GDAL is a translator library for raster and vector geospatial data formats that is released under an MIT style Open Source *License* by the Open Source Geospatial Foundation. As a library, it presents a single raster abstract data model and single vector abstract data model to the calling application for all supported formats. It also comes with a variety of useful command line utilities for data translation and processing. The NEWS page describes the June 2026 GDAL/OGR 3.13.1 release.

See *Software using GDAL*

You may quote GDAL in publications by using the following Digital Object Identifier: DOI 10.5281/zenodo.5884351

CHAPTER

TWO

LICENSE

2.1 License

In general, the GDAL/OGR source code is licensed under the MIT license, whose terms are:

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Some files are licensed under BSD 2-clause, BSD 3-clause or other non-copyleft licenses. The full licensing terms are available in the LICENSE.TXT file.

Note however that GDAL can be built against many dependencies, each of them with their own licensing terms (possibly LGPL, GPL or proprietary), hence the use of a GDAL binary can be subject to less permissive licensing terms than MIT, and it is the responsibility of users to check that they comply with the overall licensing terms.

CHAPTER

THREE

DOWNLOAD

The GDAL project distributes GDAL as source code and *Containers* only. *Binaries* produced by others are available for a variety of platforms and package managers.

3.1 Source Code

3.1.1 Current Release

- **2026-06-05** gdal-3.13.1.tar.gz 3.13.1 Release Notes (3.13.1 md5, 3.13.1 sig)

The GPG signing key is:


```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBGjlDXoBEACjbYUouwHnW+X/xK51AfwN1vTrJn+pZziGGLobD4Jfs4gU4PwQ
dEw1bFj6tsgBrJ7hrKmcyd0bBQiKU1FTYy+tOu768t9qlC7HC3JfRBACrGn3Svz4
evinQkq5GGcZ1FkJhRKgEIxJLGppG7O45XEFYdcbCkLqC8uyu2+PbY2SYXlzlKs8
DLiAwqX670eZ78pDtynJj81CRowSqyNBOzaltR6OLCsYZrYzAYYtBqSiifKoIpm/
sV4VRvUuWaaUNNMPPRLt+mV+iIYm8cKLH2uyozZWIfOSck7I30fGIuuUhulc41bi
3WNV+QtOepPyuMolab8JNbaRJXAFxQ5Q2k5ZyzXvpuAEqwrhaFIroc7LvkQ4RLlT
r6BkFUzvCCjRWG5jCFA0CjvvvYG9DPSUKdSaD2Vxfor/cwSRqnPYwRauFMuST57H
exrLYJZqfeZuqV2mlI8wVvh3tO9oC/pGmggV3B1KxYvzj2I/nQ/bnugveEvD005j
EcfSlmAPWcuily+rgB9xqed0tZwG0A9fjY/gveqxLVE41EIoAfkdeRl+KVOqEEYk
cyiJQdFKBuGxemEP2cfDnROUawp+1XXEuA1MGkLvB+6nNIp1e+mhUtWfDY70MEz1
l7J49MUZBfPM3vy63t7ihlPLIP9IUcO9j9rfy1iW8t86qG3CnZE/rsnVZwARAQAB
tCRHREFMIFJlbGVhc2UgQm90IDxnZGFsb3JnQGdtYWlsLmNvbT6JAlQEEwEKAD4W
IQR7Ew6VW0TYfM4nZfDb2B/1LsKkKgUCaOUNegIbAwUJA8JnAAULCQgHAgYVCgkI
CwIEFgIDAQIeAQIXgAAKCRDb2B/1LsKkKpY+D/9hVR9JIQNT3So4fg/fQx36azSf
kVzO4nS0lZhSxR4/Xig5Zqq8EOFk7ipWwb4hhivPi7yR49BhVSQbHo36+jNbC5k/
jDMsZ+9CNzUjp1XSH/v06mcdSD4LB4vSvHBfLjDdV7OoOGnExkEyqBcoje53tW6p
tKQGwqZ3ilZs75GkLO1UC3DNQ9VpVc7lk1Cv+npto86A6CWMAC7q+n540wC/oTj7
APn6dz1S0LAPWR9v0aBoLMoRP9YiRQSmQmyQsKe3HJd4c7hSdWyST/PIMvPZawpP
BYG+JRRuRoWm6zjJIWnvqj9IZeFkQv1fKqYHKW86aITTqGGF6kcq83mf2XH26KQR
mu8ArI5aLULmpKA67p2RXxyPdoSNiaiP8UuWQ18rnEA/Hedz2sgBYSvjeYYziTRm
wgiRbts4MjB5pj+JGKtX7FEL6HuPDVnMYGc8n8bO8qiKctuYMwBM/KRE+2e3zx0m
m5YXjjCSvBiTLYNhvrrQ5pxiUK5xiAH0G/MzPXjJToJzLrHn0vqRVHXs9F8d9zLe
o6E4S8VVgyJ9zdXkYtj9nyL9jb00K+jj/g4avpnOKTuIgpsTR2ntxWz2liWU6Saz
03Vrd9mVwgU7NBnYAMXOg54eYwhyWkGvbFHvCK6ijL1+SaAkN9mVs5VTJMpGiJ9s
qj7DbIY0GwZQMVSvxQ==
=FY/p
-----END PGP PUBLIC KEY BLOCK-----
```

3.1.2 Past Releases

Links to past releases are also available.

3.1.3 Development Source

The main repository for GDAL is located on GitHub at https://github.com/OSGeo/GDAL.

You can obtain a copy of the active source code by issuing the following command

```
git clone https://github.com/OSGeo/GDAL.git
```

Additional information is available about build_requirements and building_from_source.

3.1.4 Maintenance policy

The GDAL upstream team only maintains the branch on which the latest release has been done, with bugfixes releases issued roughly every 2 months. So, for example, during the development phase of GDAL 3.10.0, GDAL 3.9.x bugfixes releases are done based on the release/3.9 branch, but not older branches (GDAL 3.8.x or older).

3.2 Binaries

In this section we list a number of the binary distributions of GDAL all of which should have fully reproducible open source build recipes.

Note that the maintainers of those distributions are generally not the maintainers of the GDAL sources, so please report any issue specific to those builds through their own support channels.

3.2.1 Windows

Windows builds are available via Conda Forge (64-bit only). See the *Conda* or *pixi* section for more detailed information. GDAL is also distributed by GISInternals and OSGeo4W and through the NuGet and *vcpkg* package managers.

3.2.2 Linux

Packages are available for Debian, Alpine, Fedora, and other distributions.

3.2.3 Mac OS

GDAL packages are available on Homebrew.

3.2.4 Android

GDAL can be installed using *vcpkg*. You may also refer to vcpkg Android support for general instructions.

For example to install default configuration for the `arm64-android` target:

```
git clone https://github.com/Microsoft/vcpkg.git
cd vcpkg
./bootstrap-vcpkg.sh  # ./bootstrap-vcpkg.bat for Windows
./vcpkg integrate install
export ANDROID_NDK_HOME=/path/to/android_ndk_home  # to adapt
./vcpkg search gdal --featurepackages  # list optional features
./vcpkg install gdal:arm64-android  # install with default configuration
./vcpkg install gdal[poppler,netcdf]:arm64-android  # install with Poppler and netdf␣
↪support
```

3.2.5 Cross-Platform Package Managers

3.2.5.1 Conda

Conda can be used on multiple platforms (Windows, macOS, and Linux) to install software packages and manage environments. Conda packages for GDAL are available through conda-forge.

```
conda install [-c channel] [package...]
```

GDAL is available as several subpackages:

- `gdal`: Python bindings and Python utilities (depends on libgdal-core)
- `libgdal`: meta-package gathering all below libgdal-* packages (except libgdal-arrow-parquet)
- `libgdal-arrow-parquet`: *(Geo)Arrow IPC File Format / Stream* and *(Geo)Parquet* drivers as a plugin (depends on libgdal-core)
- `libgdal-avif`: *AVIF -- AV1 Image File Format* driver as a plugin (depends on libgdal-core, available since GDAL 3.10.0)
- `libgdal-core`: core library and C++ utilities, with a number of builtin drivers (available since GDAL 3.9.1)
- `libgdal-fits`: *FITS -- Flexible Image Transport System* driver as a plugin (depends on libgdal-core, available since GDAL 3.9.1)
- `libgdal-grib`: *GRIB -- WMO General Regularly-distributed Information in Binary form* driver as a plugin (depends on libgdal-core, available since GDAL 3.9.1)
- `libgdal-hdf4`: *HDF4 -- Hierarchical Data Format Release 4 (HDF4)* driver as a plugin (depends on libgdal-core, available since GDAL 3.9.1)
- `libgdal-hdf5`: *HDF5 -- Hierarchical Data Format Release 5 (HDF5)* driver as a plugin (depends on libgdal-core, available since GDAL 3.9.1)
- `libgdal-heif`: *HEIF / HEIC -- ISO/IEC 23008-12 High Efficiency Image File Format* driver as a plugin (depends on libgdal-core, available since GDAL 3.10.0)
- `libgdal-jp2openjpeg`: *JP2OpenJPEG -- JPEG2000 driver based on OpenJPEG library* driver as a plugin (depends on libgdal-core, available since GDAL 3.9.1)
- `libgdal-kea`: *KEA* driver as a plugin (depends on libgdal-core, available since GDAL 3.9.1)
- `libgdal-netcdf`: *NetCDF: Network Common Data Form* driver as a plugin (depends on libgdal-core, available since GDAL 3.9.1)
- `libgdal-pdf`: *PDF -- Geospatial PDF* driver as a plugin (depends on libgdal-core, available since GDAL 3.9.1)
- `libgdal-postgisraster`: *PostGISRaster -- PostGIS Raster driver* driver as a plugin (depends on libgdal-core, available since GDAL 3.9.1)
- `libgdal-pg`: *PostgreSQL / PostGIS* driver as a plugin (depends on libgdal-core, available since GDAL 3.9.1)
- `libgdal-tiledb`: *TileDB - TileDB raster* driver as a plugin (depends on libgdal-core, available since GDAL 3.9.1)
- `libgdal-xls`: *XLS - MS Excel format* driver as a plugin (depends on libgdal-core, available since GDAL 3.9.1)

For background on the GDAL package split on conda-forge, see Introducing Lightweight Versions of GDAL and PDAL.

To install the `gdal` package (Python bindings and utilities), and `libgdal-core`:

```
conda install -c conda-forge gdal
```

To install the `libgdal` meta-package with all available drivers, but libgdal-arrow-parquet:

```
conda install -c conda-forge libgdal
```

To install the Arrow and Parquet drivers as plugins:

```
conda install -c conda-forge libgdal-arrow-parquet
```

GDAL master Conda builds

GDAL master builds are available in the gdal-master channel. They are based on dependencies from the `conda-forge` channel. The latest master build can be installed with the following command:

```
conda install -c gdal-master -c conda-forge gdal-master::gdal
```

As with released versions of GDAL, additional drivers can be installed using `gdal-master::libgdal-{driver_name}`.

If you already have a GDAL master build installed in an environment, you can update it to the latest master version using the commands below. GDAL subpackages may need to be updated individually, because the master version numbers (e.g. `3.12.99`) do not change with every nightly build.

```
# check currently installed GDAL components and versions
conda list gdal
gdal --version
# remove current master version
conda remove gdal libgdal-core --yes
# install latest GDAL master build
conda install -c gdal-master -c conda-forge gdal libgdal-core --yes
gdal --version
```

3.2.5.2 pixi

Pixi is a package management tool for developers. It allows the developer to install libraries and applications in a reproducible way. Packages for GDAL are available through the conda-forge channel.

If you want to be able to use GDAL as part of a project:

```
pixi init name-of-project
cd name-of-project
pixi add gdal libgdal-core
pixi add libgdal-arrow-parquet # if you need the Arrow and Parquet drivers
pixi shell
```

Pixi supports using tools like GDAL and OGR globally, similar to conda's base environment, without having to use an activate command:

```
pixi global install gdal libgdal-core
pixi global install libgdal-arrow-parquet # if you need the Arrow and Parquet drivers
```

3.2.5.3 vcpkg

The GDAL port in the vcpkg dependency manager is kept up to date by Microsoft team members and community contributors. You can download and install GDAL using the vcpkg as follows:

```
git clone https://github.com/Microsoft/vcpkg.git
cd vcpkg
./bootstrap-vcpkg.sh  # ./bootstrap-vcpkg.bat for Windows
./vcpkg integrate install
./vcpkg search gdal --featurepackages  # list optional features
./vcpkg install gdal  # install with default configuration
./vcpkg install gdal[poppler,netcdf]  # install with Poppler and netdf support
```

If the version is out of date, please create an issue or pull request on the vcpkg repository.

3.2.5.4 Spack

Spack is a package management tool designed to support multiple versions and configurations of software on a wide variety of platforms and environments. It was designed for large supercomputing centers. Spack builds packages from sources, and allows tweaking their configurations.

You can find information about GDAL in Spack at https://packages.spack.io/package.html?name=gdal

For the default GDAL build with a reduced number of drivers:

```
git clone -c feature.manyFiles=true https://github.com/spack/spack.git
cd spack/bin
./spack install gdal
```

For a build with netcdf driver enabled:

```
./spack install gdal +netcdf
```

3.3 Containers

Docker images with nightly builds of GDAL master and tagged releases are available at GitHub Container registry.

Information on the content of the different configurations can be found at https://github.com/OSGeo/gdal/tree/master/docker.

3.4 Documentation

This documentation is also available as a PDF for offline browsing.

CHAPTER

FOUR

FAQ

4.1 What does GDAL stand for?

GDAL - Geospatial Data Abstraction Library

It is sometimes pronounced "goo-doll" (a bit like goo-gle), while others pronounce it "gee-doll," and others pronounce it "gee-dall."

Listen how Frank Warmerdam pronounces it and the history behind the acronym.

4.2 What is this OGR stuff?

OGR used to be a separate vector IO library inspired by OpenGIS Simple Features which was separated from GDAL. With the GDAL 2.0 release, the GDAL and OGR components were integrated together.

4.3 What does OGR stand for?

See OGR.

4.4 What does CPL stand for?

See CPL.

4.5 What does VSI stand for?

See VSI.

4.6 When was the GDAL project started?

In late 1998, Frank Warmerdam started to work as independent professional on the GDAL/OGR library.

4.7 Is GDAL/OGR proprietary software?

No, GDAL/OGR is a Free and Open Source Software.

4.8 What license does GDAL/OGR use?

See *License*

4.9 What operating systems does GDAL-OGR run on?

You can use GDAL/OGR on all modern flavors of Unix: Linux, FreeBSD, Mac OS X; all supported versions of Microsoft Windows; mobile environments (Android and iOS). Both 32-bit and 64-bit architectures are supported.

4.10 Is there a graphical user interface to GDAL/OGR?

Plenty! Among the available options, QGIS is an excellent free and open source application with a rich graphical user interface. It allows users to display most GDAL raster and vector supported formats and provides access to most GDAL utilities through its Processing toolbox.

You can also consult the *list of software using GDAL*.

4.10.1 Software using GDAL

4.10.1.1 Free and open source

- actinia A cloud based geoprocessing platform from mundialis (GPL v3)
- Biodiverse A tool for the spatial analysis of diversity. Uses GDAL for import/export of data (GPL v3)
- Bluemapia Multi-Map(Google,Microsoft,Open Street Map, NOAA/BSB Charts,self-calibrated raster) location-based GPS app for Windows Mobile. (GPL v2)
- BRL-CAD An open source solid modeling computer-aided design system.
- Demeter Another OpenGL based terrain engine somewhat similar to VTP.
- DuckDB Spatial Extension: Spatial extension for DuckDB
- EOxServer OGC-compliant server for Earth Observation (EO) data supporting WMS and WCS with EO application profiles (EOxServer Open License)
- exactextract Fast and accurate raster/vector zonal statistics (Apache 2.0)
- Feature Data Objects (FDO) Open source spatial data access libraries (LGPL)
- Fiona Fiona is OGR's neater Python API – sleek and elegant on the outside, indomitable power on the inside.
- gdal3.js.org Convert raster and vector geospatial data to various formats and coordinate systems entirely in the browser.
- *GdalToTiles* C# Program (open source) for making image tiles for Google Earth with KML Superoverlay. (project discontinued; was hosted on CodePlex which shut down in December 2017)
- GeoDa Introduction to Spatial Data Analysis (spatial autocorrelation and spatial regression) (GPL)
- GeoDjango A framework for building geographic web applications.
- geofileops Python toolbox to process large vector files faster.
- GeoKettle An open source spatial ETL (Extract, Transform and Load) tool (LGPL)
- GeoNotebook a Jupyter notebook extension for geospatial visualization and analysis. (Apache 2.0)
- GeoServer a open source software server written in Java that allows users to share and edit geospatial data.

- GMT (Generic Mapping Tools) an open source collection of tools for processing and displaying xy and xyz datasets.
- GRASS GIS A raster/vector open source GIS that uses GDAL for raster/vector import and export (via r.in.gdal/r.out.gdal)
- gstat a geostatistical modelling package.
- GuidosToolbox A multi-platform desktop application for generic image object analysis.
- gvSIG Desktop GIS Client.
- ILWIS Remote Sensing and GIS Desktop Package.
- Image I/O-Ext includes gdalframework, a framework leveraging on GDAL via SWIG's generated JAVA bindings to provide support for a broad set of data formats.
- libLAS Open Source LAS 1.0/1.1 ASPRS LiDAR data translation toolset
- Loader A simple loader for geographic data in GML and KML that needs some preparation before loading via ogr2ogr.
- MapGuide Open source web mapping server.
- Mapnik C++/ Python mapping toolkit
- MapServer A popular web mapping application with GDAL support.
- MapWindow open source ActiveX control with GIS functionality.
- Mergin Maps Field data collection app built on top of QGIS with cloud sync.
- MultiSpec Application for interactively analyzing multispectral/hyperspectral image data.
- NASA Ames Stereo Pipeline Software for creating terrain models and ortho images from planetary stereo images. (Apache 2.0)
- NASA WorldWind Multiplatform virtual globe library to quickly and easily create interactive visualizations of 3D globes, map and geographical information.
- NextGIS Formbuilder Desktop application for creating and editing forms.
- NextGIS Web Server-side Web GIS and a framework for storage, visualization and permissions management of all kinds
- Ogr2 GUI Graphical user interface for ogr2ogr
- OpenCPN A concise ChartPlotter/Navigator. A cross-platform ship-borne GUI application.
- OpenDataCube FOSS software package that simplifies the management and analysis of large amounts of satellite imagery and other Earth observation data.
- OpenEV An OpenGL/GTK/Python based graphical viewer which exclusively uses GDAL for raster access.
- OFGT a collection of utilities for multipurpose forest monitoring under the Open Foris Initiative Open Foris Initiative.
- OpenFLUID a software platform for spatial modelling of landscapes dynamics
- OpenSceneGraph 3D rendering engine with osgdem and osgEarth plugins.
- Opticks an open source remote sensing application and development framework, with a GDAL plugin.
- Orfeo Toolbox (OTB) a general remote sensing image processing library.
- OSSIM Another geospatial viewing and analysis environment which uses GDAL as one of several plugins.
- PDAL Point Cloud Data Abstraction Library

- pktools open source (GPLv3) tools written in C++ for remote sensing image processing
- PNMapcalc A raster map calculator with C-like scripting language.
- PostGIS spatial database extender for PostgreSQL: The raster loader and many of the raster SQL functions rely on GDAL.
- PostgreSQL OGR Foreign Data Wrapper Expose OGR layer as PostgreSQL foreign tables.
- QGIS A cross platform desktop GIS.
- R A free software environment for statistical computing and graphics, with bindings to GDAL via the rgdal package.
- Rasterio Python library and command line utilities to read and write GDAL rasters.
- Rasterix A cross platform open source utility to process raster data based on Qt and GDAL.
- SAGA GIS A free geographic information system (GIS), with a special 'Application Programming Interface' (API) for geographic data processing.
- SNAP Sentinel Application Platform for Earth Observation processing and analysis.
- StarSpan raster/vector analysis.
- t-rex Vector tile server written in Rust.
- Thuban A multi-platform interactive geographic data viewer.
- Vertual Terrain Project fostering tools for easy construction of the real world in interactive, 3D digital form.
- ViRGiS Creating a GIS Platform in Virtual Reality / Mixed Reality

4.10.1.2 Proprietary license / Other

- 3D DEM Viewer from MS MacroSystem.
- Cadcorp SIS: A Windows GIS with a GDAL and OGR plugins.
- Carmenta Engine (previously known as SpatialAce): A GIS Rapid Application Development environment
- CARTO A cloud mapping platform to analyze and visualize geospatial data.
- CatchmentSIM A Windows terrain analysis model for hydrologic applications.
- Daylon Leveller A terrain/heightfield/bumpmap modeler
- ERDAS ER Viewer Image viewer for very large JPEG 2000 and ECW files.
- ESRI ArcGIS Pro A popular GIS platform.
- Eternix Blaze Advanced geo-spatial visualization application and SDK.
- FalconView Windows-based GIS platform with roots in military mission planning, now available as a free GIS visualization and analysis package.
- flighttrack GPS track viewing and downloading software for Mac.
- FME A GIS translator package includes a GDAL plugin.
- GenGIS Software for geospatial analysis of genetic data.
- Geographic Imager DEM / aerial / satellite image processing GIS plug-in for Adobe Photoshop, by Avenza Systems.
- GeoDMS A framework for building spatial calculation models.
- GeoFusion 3D visualization.

- Geoweb3d A 3D virtual globe that provides on-the-fly, game-quality visualization of GIS data.
- Google Earth A 3D world viewer.
- HydroDaVE Explorer A web-enabled client that provides users an easy to use, secure, and reliable data management platform to efficiently manage, access, and analyze environmental data.
- IDRISI A GIS and Image Processing Windows Desktop application. Uses GDAL to import/export/warp raster data.
- Infraworks a BIM software for infrastructure project design, part of the Autodesk suite.
- iShare Web data integration and publishing platform by Astun Technology.
- Makai Voyager An advanced 3D/4D geospatial visualization platform.
- MapInfo Professional Desktop GIS and mapping application
- MapTiler Generator of tiles for interactive maps and overlays made from raster images and geodata.
- Maptitude Mapping Software Desktop GIS and business mapping application
- MicroImages TNT advanced software for geospatial analysis (Windows, Linux, Mac OS X and UNIX)
- Micromine A mining software solution that uses GDAL for reading/writing various geospatial file formats.
- Mirone Matlab based package for geospatial, oceanographic and geophysical analysis of grids
- MyGeodata Cloud Online converter of GDAL raster and OGR vector formats
- OPALS Orientation and Processing of Airborne Laser Scanning Data
- Procura Landholding inspection system developed for the UK Homes and Communities Agency. GDAL is used for checking out background mapping.
- ScanMagic Win32 application for visualization, analysis and processing of remote sensing data.
- Scalable Algorithmics (SCALGO) Software for efficiently constructing and performing computations on very large raster and TIN terrain models.
- Scenomics Software for building terrain databases uses GDAL for projection and data import/export.
- scenProc scenProc: A tool to create scenery for Microsoft Flight Simulator and Lockheed Martin Prepar3D by processing G
- SeaView A 3D GIS package for geophysical and hydrographical data (side scan sonar, subbottom profiler, magnetometer, multibeam, etc.
- SkylineGlobe The Skyline suite of interactive applications allows you to build, view, query and analyze customized, virtual 3D landscapes.
- SpacEyes3D 3D visualization software for cartographic data.
- Spatial Manager A product suite designed designed to manage spatial data in a simple, fast and inexpensive way. Uses GDAL to import/export data.
- TacitView An imagery visualization and exploitation package for military intelligence.
- TatukGIS Desktop GIS mapping and data editing application.
- Team Awareness Kit Suite of georeferenced imagery and situational awareness tools developed for military planning and execution, now available for civilian use.
- TerrainView Interactive real-time 3D GIS Software.
- TransCAD GIS Desktop Transportation Analysis Software
- TravTime .NET Application for visualizing, processing and analyzing GPS data for travel time, speed, and de

- VectorWorks The Vectorworks line of software products provides professional design solutions in the AEC, entertainment and landscape design industries.
- WindNinja wind model for fire behavior modeling.

4.11 What compiler can I use to build GDAL/OGR?

GDAL/OGR must be compiled with a C++17 capable compiler since GDAL 3.9 (C++11 in previous versions)

Build requirements are described in build_requirements.

4.12 I have a question that's not answered here. Where can I get more information?

See *Community*

Keep in mind, the quality of the answer you get does bear some relation to the quality of the question. If you need more detailed explanation of this, you can find it in essay How To Ask Questions The Smart Way by Eric S. Raymond.

4.13 How do I add support for a new format?

To some extent this is now covered by the *Raster driver implementation tutorial* and *Vector driver implementation tutorial*

4.14 What about file formats for 3D models?

While some vector drivers in GDAL support 3D geometries (e.g. reading CityGML or AIXM through the GML driver, or 3D geometries in GeoPackage), 3-dimensional scenes or models formats (such as glTF, OBJ, 3DS, etc.) are out-of-scope for GDAL. Users may consider a library such as ASSIMP (Open-Asset-Importer-Library) for such formats.

4.15 Is GDAL thread-safe ?

See *Multi-threading*

4.16 Does GDAL provide a Section 508 information?

No, GDAL itself is an open-source software and project, not a Vendor. If your organization considers they need a VPAT or Section 508 form to be able to use GDAL, it is their responsibility to complete the needed steps themselves.

4.17 How do I cite GDAL ?

See CITATION

CHAPTER

FIVE

PROGRAMS

5.1 "gdal" application

Added in version 3.11.

Starting with GDAL 3.11, parts of the GDAL utilities are available from a new single **gdal** program that accepts commands and subcommands.

As an introduction, you can follow the webinar given on June 3, 2025 about the GDAL Command Line Interface Modernization as a PDF slide deck or the recording of the video.

> **Warning**
>
> The **gdal** command is provisionally provided as an alternative interface to GDAL and OGR command line utilities. The project reserves the right to modify, rename, reorganize, and change the behavior of the utility until it is officially frozen via PSC vote in a future major GDAL release. The utility needs time to mature, benefit from incremental feedback, and explore enhancements without carrying the burden of full backward compatibility. Your usage of it should have no expectation of compatibility until that time.

5.1.1 General

5.1.1.1 Main gdal entry point

Added in version 3.11.

Synopsis

```
Usage: gdal <COMMAND> [OPTIONS]
where <COMMAND> is one of:
  - convert:  Convert a dataset (shortcut for 'gdal raster convert' or 'gdal vector␣
↪convert').
  - dataset:  Commands to manage datasets.
  - driver:   Command for driver specific operations.
  - info:     Return information on a dataset (shortcut for 'gdal raster info' or 'gdal␣
↪vector info').
  - mdim:     Multidimensional commands.
  - pipeline: Process a dataset applying several steps.
  - raster:   Raster commands.
  - vector:   Vector commands.
  - vsi:      GDAL Virtual System Interface (VSI) commands.
```

(continues on next page)

(continued from previous page)

```
Common Options:
  -h, --help              Display help message and exit
  --json-usage            Display usage as JSON document and exit
  --config <KEY>=<VALUE>  Configuration option [may be repeated]

Options:
  --version               Display GDAL version and exit
  --drivers               Display driver list as JSON document

'gdal <FILENAME>' can also be used as a shortcut for 'gdal info <FILENAME>'.
And 'gdal read <FILENAME> ! ...' as a shortcut for 'gdal pipeline <FILENAME> ! ...'.
```

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Display the GDAL version

```
$ gdal --version
GDAL 3.14.0dev-687b2cae6c, released 2026/06/05 (debug build)
```

Example 2: Getting information on the file utm.tif (with JSON output)

```
$ gdal info utm.tif
```

Example 3: Converting file utm.tif to GeoPackage raster

```
$ gdal convert utm.tif utm.gpkg
```

Example 4: Getting information on all available commands and subcommands as a JSON document.

```
$ gdal --json-usage
```

Example 5: Getting list of all formats supported by the current GDAL build, as text

```
$ gdal --formats
```

Example 6: Search for Parquet in the list of all formats using jq

Bash

PowerShell

```
$ gdal --drivers | jq '.[] | select(.short_name == "Parquet")'
```

```
gdal --drivers | jq '.[] | select(.short_name == \"Parquet\")'
```

5.1.1.2 Syntax for commands of `gdal` program

Added in version 3.11.

- Commands accept one or several positional arguments, typically for dataset names. The order is input(s) first, output last.

  Positional arguments can also be specified as named arguments, if preferred to avoid any ambiguity.

- Named arguments have:

  - at least one "long" name, preceded by two dash characters when specified on the command line,
  - optionally auxiliary long names,
  - and optionally a one-letter short name, preceded by a single dash character on the command line.

  e.g. `-f, --of, --format, --output-format <OUTPUT-FORMAT>`

  Boolean arguments (also called flags) are enabled by specifying the argument name without a value.

  e.g. `--overwrite`

  Arguments that require a value are specified like:

  - `-f VALUE` for one-letter short names.
  - `--format VALUE` or `--format=VALUE` for long names.

  Some arguments can be multi-valued. Some of them require all values to be packed together and separated by commas. This is, for example, the case of `--bbox <BBOX> Clipping bounding box as xmin,ymin, xmax,ymax`. e.g. `--bbox=2.1,49.1,2.9,49.9`

  Others accept each value to be preceded by a new mention of the argument name. e.g `--co COMPRESS=LZW --co TILED=YES`. In this example, if the value of the argument does not contain commas, the packed form is also accepted: `--co COMPRESS=LZW,TILED=YES`.

- Named arguments can be placed before or after positional arguments.

Example

Given the following (simplified) synopsis:

```
gdal raster convert [OPTIONS] <INPUT> <OUTPUT>

Convert a raster dataset.

Positional arguments:
  -i, --input <INPUT>                                    Input raster dataset [required]
  -o, --output <OUTPUT>                                  Output raster dataset (created by␣
↪algorithm) [required]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]
  --overwrite                                            Whether overwriting existing␣
↪output is allowed
```

The following command lines are valid:

```
gdal raster convert input.tif output.tif
gdal raster convert -i input.tif -o output.tif
gdal raster convert --input=input.tif --output=output.tif
gdal raster convert --input input.tif --output output.tif
gdal raster convert --creation-option COMPRESS=LZW --creation-option TILED=YES --
↪overwrite input.tif output.tif
gdal raster convert input.tif output.tif --co COMPRESS=LZW,TILED=YES --overwrite
```

Working with JSON Output

Several GDAL commands can produce JSON output. jq is a useful tool to parse this JSON and extract specific information. Precompiled binaries of jq are available for Linux, macOS, and Windows.

Some example workflows are listed below:

- *Search for Parquet in the list of all formats using `jq`*
- *List all layers and their geometry fields using `jq`*

Suggestions for argument values

As an alternative to *Bash completion for gdal*, it is possible to ask for potential enumerated values for certain arguments by appending =? to the argument name. This feature is available in all shells and on all operating systems.

Example 1: Asking for the layer names of a vector dataset

```
gdal vector info my.gpkg --layer=?
```

can return:

```
ERROR 1: info: Potential values for argument 'layer' are:
- towns
- countries
```

Example 2: Asking for the allowed values of the resampling argument

```
gdal raster reproject --resampling=?
```

returns:

```
ERROR 1: reproject: Potential values for argument 'resampling' are:
- nearest
- bilinear
- cubic
- cubicspline
- lanczos
- average
- rms
- mode
- min
- max
- med
- q1
```

(continues on next page)

(continued from previous page)

```
- q3
- sum
```

5.1.1.3 Migration guide to gdal command line interface

This page documents, through examples, how to migrate from the traditional GDAL command line utilities to the unified gdal command line interface added in GDAL 3.11.

Raster commands

- Getting information on a raster dataset in human-readable format

```
gdalinfo my.tif

==>

gdal raster info my.tif
```

- Converting a georeferenced netCDF file to cloud-optimized GeoTIFF

```
gdal_translate -of COG in.nc out.tif

==>

gdal raster convert --of=COG in.nc out.tif
```

- Reprojecting a GeoTIFF file to a Deflate compressed tiled GeoTIFF file

```
gdalwarp -t_srs EPSG:4326 -co TILED=YES -co COMPRESS=DEFLATE -overwrite in.tif out.tif

==>

gdal raster reproject --output-crs=EPSG:4326 --co=TILED=YES,COMPRESS=DEFLATE --overwrite␣
↪in.tif out.tif
```

- Update existing out.tif with content of in.tif using cubic interpolation

```
gdalwarp -r cubic in.tif out.tif

==>

gdal raster update -r cubic in.tif out.tif
```

- Converting a PNG file to a tiled GeoTIFF file, adding georeferencing for world coverage in WGS 84 and metadata

```
gdal_translate -a_ullr -180 90 180 -90 -a_srs EPSG:4326 -co TILED=YES -mo␣
↪DESCRIPTION=Mean_day_temperature in.png out.tif

==>

gdal raster pipeline read in.png ! edit --crs=EPSG:4326 --bbox=-180,-90,180,90 --
↪metadata=DESCRIPTION=Mean_day_temperature ! write --co=TILED=YES out.tif
```

Note that the order of elements differ: "upper-left-x upper-left-y lower-right-x lower-right-y" for gdal_translate, compared to "minimum-x,minimum-y,maximum-x,maximum-y" for the `--bbox` option of "gdal raster pipeline ... edit".

- Clipping a raster with a bounding box

```
gdal_translate -projwin 2 50 3 49 in.tif out.tif

==>

gdal raster clip --bbox=2,49,3,50 in.tif out.tif
```

- Creating a virtual mosaic (.vrt) from all GeoTIFF files in a directory

```
gdalbuildvrt out.vrt src/*.tif

==>

gdal raster mosaic src/*.tif out.vrt
```

- Creating a mosaic in COG format from all GeoTIFF files in a directory

```
gdalbuildvrt tmp.vrt src/*.tif
gdal_translate -of COG tmp.vrt out.tif

==>

gdal raster mosaic --of=COG src/*.tif out.tif
```

- Adding internal overviews for reduction factors 2, 4, 8 and 16 to a GeoTIFF file

```
gdaladdo -r average my.tif 2 4 8 16

==>

gdal raster overview add -r average --levels=2,4,8,16 my.tif
```

- Combining single-band rasters into a multi-band raster

```
gdalbuildvrt tmp.vrt red.tif green.tif blue.tif
gdal_translate tmp.vrt out.tif

==>

gdal raster stack red.tif green.tif blue.tif out.tif
```

- Reorder a 3-band dataset with bands ordered Blue, Green, Red to Red, Green, Blue

```
gdal_translate -b 3 -b 2 -b 1 bgr.tif rgb.tif

==>

gdal raster select --band 3,2,1 bgr.tif rgb.tif --overwrite
```

- Expand a dataset with a color table to RGB

```
gdal_translate -expand rgb color_table.tif rgb.tif

==>

gdal raster color-map color_table.tif rgb.tif --overwrite
```

- Apply an external color-map to a dataset

```
gdaldem color-map color_table.tif color_map.txt rgb.tif

==>

gdal raster color-map --color-map=color_map.txt color_table.tif rgb.tif --overwrite
```

- Convert nearly black values of the collar to black

```
nearblack -nb 1 -near 10 my.tif

==>

gdal raster clean-collar --update --color-threshold=1 --pixel-distance=10 my.tif
```

- Generating tiles between zoom level 2 and 5 of WebMercator from an input GeoTIFF

```
gdal2tiles --zoom=2-5 input.tif output_folder

==>

gdal raster tile --min-zoom=2 --max-zoom=5 input.tif output_folder
```

Vector commands

- Getting information on a vector dataset in human-readable format

```
ogrinfo -al -so my.gpkg

==>

gdal vector info my.gpkg
```

- Converting a shapefile to a GeoPackage

```
ogr2ogr out.gpkg in.shp

==>

gdal vector convert in.shp out.gpkg
```

- Reprojecting a shapefile to a GeoPackage

```
ogr2ogr -t_srs EPSG:4326 out.gpkg in.shp

==>
```

(continues on next page)

(continued from previous page)

```
gdal vector reproject --output-crs=EPSG:4326 in.shp out.gpkg
```

- Clipping a GeoPackage file

```
ogr2ogr -clipsrc 2 49 3 50 out.gpkg in.shp

==>

gdal vector clip --bbox=2,49,3,50 in.gpkg out.gpkg
```

- Selecting features from a GeoPackage file intersecting a bounding box, but not clipping them to it

```
ogr2ogr -spat 2 49 3 50 out.gpkg in.shp

==>

gdal vector filter --bbox=2,49,3,50 in.gpkg out.gpkg
```

- Selecting features from a shapefile intersecting a bounding box, but not clipping them to it and reprojecting

```
ogr2ogr -t_srs EPSG:32631 -spat 2 49 3 50 out.gpkg in.shp

==>

gdal vector pipeline read in.gpkg ! filter --bbox=2,49,3,50 ! reproject --output-
↪crs=EPSG:32631 ! write out.gpkg
```

- Selecting features from a shapefile based on an attribute query, and restricting to a few fields

```
ogr2ogr -where "country='Greenland'" -select population,_ogr_geometry_ out.gpkg in.shp

==>

gdal vector pipeline ! read in.shp ! filter --where "country='Greenland'" ! select --
↪fields population,_ogr_geometry_ ! write out.gpkg
```

- Creating a GeoPackage stacking all input shapefiles in separate layers.

```
ogrmerge -f GPKG -o merged.gpkg *.shp

==>

gdal vector concat --mode=stack *.shp merged.gpkg
```

- Modify in-place a GeoPackage dataset by running a SQL command.

```
ogrinfo my.gpkg -sql "DELETE FROM countries WHERE pop > 1e6"

==>

gdal vector sql --update my.gpkg --sql "DELETE FROM countries WHERE pop > 1e6"
```

Python commands

The following commands were previously provided as individual *Python scripts*, but are now accessible through the unified "gdal" command-line interface.

- Listing the contents of a remote *PMTiles* dataset, using *gdal vsi list*.

```
python gdal_ls.py -lr "/vsipmtiles//vsicurl/https://protomaps.github.io/PMTiles/
↪protomaps(vector)ODbL_firenze.pmtiles"

==>

gdal vsi list -lR "/vsipmtiles//vsicurl/https://protomaps.github.io/PMTiles/
↪protomaps(vector)ODbL_firenze.pmtiles"
```

- Listing the contents of a vsis3 dataset.

```
export AWS_NO_SIGN_REQUEST="YES"
gdal_ls.py "/vsis3/overturemaps-us-west-2/release/2025-10-22.0/theme=buildings/
↪type=building"

==>

export AWS_NO_SIGN_REQUEST="YES"
gdal vsi list "/vsis3/overturemaps-us-west-2/release/2025-10-22.0/theme=buildings/
↪type=building"
```

- Extracting all remote content into a local directory, using *gdal vsi copy*.

```
gdal_cp.py "/vsipmtiles//vsicurl/https://protomaps.github.io/PMTiles/
↪protomaps(vector)ODbL_firenze.pmtiles/metadata.json" /vsistdout/ | jq

==>

gdal vsi copy -r "/vsipmtiles//vsicurl/https://protomaps.github.io/PMTiles/
↪protomaps(vector)ODbL_firenze.pmtiles" out_pmtiles
```

5.1.1.4 Bash completion for `gdal`

Added in version 3.11.

If Bash is used as an interactive shell, users can benefit from completion, using the <TAB> key. Bash completion is enabled by first sourcing the `$install_prefix/share/bash-completion/completions/gdal` file, which should be automatically done on a number of binary distributions of GDAL.

That script can also be sourced when using **zsh** as a shell (starting with GDAL 3.11.1)

Examples of completion

Example 1: Listing sub-commands of "gdal":

```
$ gdal <TAB><TAB>
==>
convert   info      pipeline  raster    vector
```

Example 2: Completion of a sub-command from its initial letters:

```
$ gdal r<TAB>
==>
$ gdal raster
```

Example 3: Listing sub-commands of "gdal raster":

```
$ gdal raster<TAB><TAB>
==>
convert    edit       info       pipeline   reproject
```

Example 4: Listing switches of "gdal raster"

```
$ gdal raster -<TAB><TAB>
==>
--approx-stats   -f               --help           --if             --json-usage     --
↪min-max         --no-fl          --no-md          --oo             --stats
--checksum       --format         --hist           --input          --list-mdd       --
↪mm              --no-gcp         --no-nodata      --open-option    --subdataset
--drivers        -h               -i               --input-format   --mdd            --
↪no-ct           --no-mask        --of             --output-format  --version
```

Example 5: Listing allowed values for a switch

```
$ gdal raster info --of=<TAB><TAB>
==>
json  text
```

Example 6: Listing allowed creation options, restricted to those valid for the output format, once the output filename has been specified

```
$ gdal raster convert in.tif out.tif --co <TAB><TAB>
==>
ALPHA=                           ENDIANNESS=                      JXL_EFFORT=           ␣
↪          PIXELTYPE=                       SOURCE_PRIMARIES_RED=            TIFFTAG_
↪TRANSFERRANGE_BLACK=
BIGTIFF=                         GEOTIFF_KEYS_FLAVOR=             JXL_LOSSLESS=         ␣
↪          PREDICTOR=                       SOURCE_WHITEPOINT=               TIFFTAG_
↪TRANSFERRANGE_WHITE=
BLOCKXSIZE=                      GEOTIFF_VERSION=                 LZMA_PRESET=          ␣
↪          PROFILE=                         SPARSE_OK=                       TILED=
[ ... snip ... ]
```

Example 7: Listing known configuration options starting with AWS

```
$ gdal --config AWS_<TAB><TAB>
==>
AWS_ACCESS_KEY_ID=                       AWS_DEFAULT_REGION=                      AWS_
```

(continues on next page)

(continued from previous page)

```
↪REQUEST_PAYER=                        AWS_STS_ENDPOINT=
AWS_CONFIG_FILE=                         AWS_HTTPS=                              AWS_
↪ROLE_ARN=                             AWS_STS_REGION=
AWS_CONTAINER_AUTHORIZATION_TOKEN=       AWS_MAX_KEYS=                           AWS_
↪ROLE_SESSION_NAME=                    AWS_STS_REGIONAL_ENDPOINTS=
AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE=  AWS_NO_SIGN_REQUEST=                    AWS_S3_
↪ENDPOINT=                           AWS_TIMESTAMP=
AWS_CONTAINER_CREDENTIALS_FULL_URI=      AWS_PROFILE=                            AWS_
↪SECRET_ACCESS_KEY=                    AWS_VIRTUAL_HOSTING=
AWS_DEFAULT_PROFILE=                     AWS_REGION=                             AWS_
↪SESSION_TOKEN=                        AWS_WEB_IDENTITY_TOKEN_FILE=
```

Example 8: Auto-completion of EPSG CRS codes

```
$ gdal raster reproject --output-crs EPSG:432<TAB>
==>
4322 -- WGS 72                  4324 -- WGS 72BE                4326 -- WGS 84          ␣
↪       4327 -- WGS 84 (geographic 3D)  4328 -- WGS 84 (geocentric)     4329 -- WGS 84␣
↪(3D)
```

Example 9: Auto-completion of filenames in a cloud storage (assuming credentials are properly set up)

```
$ gdal raster info /vsis3/my_bucket/b<TAB><TAB>
==>
/vsis3/my_bucket/byte.tif      /vsis3/my_bucket/byte2.tif
```

5.1.1.5 How to use `gdal` CLI algorithms from C

Added in version 3.12.

The C API for `gdal` CLI algorithms is available in `gdalalgorithm.h`.

The first step is to get the instance of the `GDALGlobalAlgorithmRegistry` with *GDALGetGlobalAlgorithmRegistry()* and then call *GDALAlgorithmRegistryInstantiateAlgFromPath()* with the path of the algorithm. Once done with a `GDALAlgorithmRegistryH` handle, the memory associated to it can be released with *GDALAlgorithmRegistryRelease()* (the algorithm handles got from the registry will still be valid).

For example:

```
GDALAlgorithmRegistryH hRegistry = GDALGetGlobalAlgorithmRegistry();
const char *const papszAlgPath[] = { "raster", "reproject", NULL };
GDALAlgorithmH hAlg = GDALAlgorithmRegistryInstantiateAlgFromPath(hRegistry,␣
↪papszAlgPath);
GDALAlgorithmRegistryRelease(hRegistry);
```

Arguments can be queried with *GDALAlgorithmGetArg()* and set through different setters depending on the type of the argument, such as *GDALAlgorithmArgSetAsString()*, *GDALAlgorithmArgSetAsInteger()*, *GDALAlgorithmArgSetAsDouble()*, etc. Once done with the argument handle, it must be released with *GDALAlgorithmArgRelease()*.

For example:

```
{
    GDALAlgorithmArgH hArg = GDALAlgorithmGetArg(hAlg, "input");
    GDALAlgorithmArgSetAsString(hArg, "byte.tif");
    GDALAlgorithmArgRelease(hArg);
}
{
    GDALAlgorithmArgH hArg = GDALAlgorithmGetArg(hAlg, "resolution");
    const double res[] = {60, 60};
    GDALAlgorithmArgSetAsDoubleList(hArg, 2, res);
    GDALAlgorithmArgRelease(hArg);
}
```

Execution of the algorithm is performed by *GDALAlgorithmRun()*, with an optional progress function. Proper closing of the output dataset must be performed with *GDALAlgorithmFinalize()*. Once done with the algorithm handle, it must be released with *GDALAlgorithmFinalize()*.

```
int ret = 0;
if( GDALAlgorithmRun(hAlg, NULL, NULL) )
{
    // do something
    GDALAlgorithmArgRelease(hArg);
}
else
{
    fprintf(stderr, "failure\n");
    ret = 1;
}
GDALAlgorithmFinalize(hAlg);
GDALAlgorithmRelease(hAlg);
```

When outputting to a *MEM -- In Memory datasets* dataset, you typically want to get the output dataset with:

```
GDALAlgorithmArgH hArg = GDALAlgorithmGetArg(hAlg, "output");
GDALArgDatasetValueH hVal = GDALAlgorithmArgGetAsDatasetValue(hArg);
GDALDatasetH hDS = GDALArgDatasetValueGetDatasetRef(hVal);
```

Putting all things together:

```
#include "gdal.h"
#include "gdalalgorithm.h"

#include <stdio.h>

int main()
{
    GDALAllRegister();
    GDALAlgorithmRegistryH hRegistry = GDALGetGlobalAlgorithmRegistry();
    const char *const papszAlgPath[] = { "raster", "reproject", NULL };
    GDALAlgorithmH hAlg = GDALAlgorithmRegistryInstantiateAlgFromPath(hRegistry,␣
↪papszAlgPath);
    GDALAlgorithmRegistryRelease(hRegistry);
    if( !hAlg )
    {
```

(continues on next page)

(continued from previous page)

```
        fprintf(stderr, "cannot instantiate algorithm\n");
        return 1;
    }

    {
        GDALAlgorithmArgH hArg = GDALAlgorithmGetArg(hAlg, "input");
        GDALAlgorithmArgSetAsString(hArg, "byte.tif");
        GDALAlgorithmArgRelease(hArg);
    }

    {
        GDALAlgorithmArgH hArg = GDALAlgorithmGetArg(hAlg, "output-format");
        GDALAlgorithmArgSetAsString(hArg, "MEM");
        GDALAlgorithmArgRelease(hArg);
    }

    {
        GDALAlgorithmArgH hArg = GDALAlgorithmGetArg(hAlg, "dst-crs");
        GDALAlgorithmArgSetAsString(hArg, "EPSG:26711");
        GDALAlgorithmArgRelease(hArg);
    }

    {
        GDALAlgorithmArgH hArg = GDALAlgorithmGetArg(hAlg, "resolution");
        const double res[] = {60, 60};
        GDALAlgorithmArgSetAsDoubleList(hArg, 2, res);
        GDALAlgorithmArgRelease(hArg);
    }

    int ret = 0;
    if( GDALAlgorithmRun(hAlg, NULL, NULL) )
    {
        GDALAlgorithmArgH hArg = GDALAlgorithmGetArg(hAlg, "output");
        GDALArgDatasetValueH hVal = GDALAlgorithmArgGetAsDatasetValue(hArg);
        GDALDatasetH hDS = GDALArgDatasetValueGetDatasetRef(hVal);
        printf("width=%d, height=%d\n", GDALGetRasterXSize(hDS),
↪GDALGetRasterYSize(hDS));
        GDALAlgorithmArgRelease(hArg);
    }
    else
    {
        fprintf(stderr, "failure\n");
        ret = 1;
    }
    GDALAlgorithmFinalize(hAlg);
    GDALAlgorithmRelease(hAlg);
    return ret;
}
```

5.1.1.6 How to use gdal CLI algorithms from C++

Added in version 3.12.

gdal CLI algorithms are available as *GDALAlgorithm* instances, and the include file to use is gdalalgorithm.h.

The first step is to get the instance of the GDALGlobalAlgorithmRegistry with *GDALGlobalAlgorithmRegistry::GetSingleton()* and then call *GDALAlgorithmRegistry::Instantiate()* with the path of the algorithm.

For example:

```
auto &singleton = GDALGlobalAlgorithmRegistry::GetSingleton();
GDALAlgorithm *algPtr = singleton.Instantiate("raster", "reproject");
if (!algPtr)
{
    std::cout << "cannot instantiate algorithm" << std::endl;
    return 1;
}
GDALAlgorithm &alg = *algPtr;
```

Arguments can be queried and set through *GDALAlgorithm::operator[]()*.

For example:

```
alg["input"] = "byte.tif";
alg["output"] = "out.tif";
alg["dst-crs"] = "EPSG:26711";
alg["resampling"] = "cubic";
alg["resolution"] = std::vector<double>{60, 60};
alg["overwrite"] = true;
```

Execution of the algorithm is performed by *GDALAlgorithm::Run()*, with an optional progress function. Proper closing of the output dataset must be performed with *GDALAlgorithm::Finalize()*

For example:

```
if (alg.Run() && alg.Finalize())
{
    std::cout << "success" << std::endl;
    return 0;
}
else
{
    std::cout << "failure" << std::endl;
    return 1;
}
```

When outputting to a *MEM -- In Memory datasets* dataset, you typically want to get the output dataset with:

```
GDALDataset *poDS = alg["output"].Get<GDALArgDatasetValue>().GetDatasetRef();
```

Putting all things together:

```
#include "gdal.h"
#include "gdalalgorithm.h"
#include "gdal_dataset.h"
```

(continues on next page)

(continued from previous page)

```
#include <iostream>

int main()
{
    GDALAllRegister();
    auto& singleton = GDALGlobalAlgorithmRegistry::GetSingleton();
    GDALAlgorithm *algPtr = singleton.Instantiate("raster", "reproject");
    if( !algPtr )
    {
        std::cout << "cannot instantiate algorithm" << std::endl;
        return 1;
    }
    GDALAlgorithm& alg = *algPtr;
    alg["input"] = "byte.tif";
    alg["output-format"] = "MEM";
    alg["dst-crs"] = "EPSG:26711";
    alg["resampling"] = "cubic";
    alg["resolution"] = std::vector<double>{60, 60};
    if( alg.Run() )
    {
        GDALDataset *poDS = alg["output"].Get<GDALArgDatasetValue>().GetDatasetRef();
        std::cout << "width=" << poDS->GetRasterXSize() << std::endl;
        std::cout << "height=" << poDS->GetRasterYSize() << std::endl;
        alg.Finalize();
        return 0;
    }
    else
    {
        std::cout << "failure" << std::endl;
        return 1;
    }
}
```

5.1.1.7 How to use `gdal` CLI algorithms from Python

Principles

`gdal` CLI algorithms are available as *`osgeo.gdal.Algorithm`* instances.

A convenient way to access an algorithm and run it is to use the *`osgeo.gdal.Run()`* function.

Run(**alg*, *arguments={}*, *progress=None*, ***kwargs*)

Run a GDAL algorithm and return it.

Added in version 3.11.

Parameters

- **alg** (*str, list[str], tuple[str],* `Algorithm`) -- Path to the algorithm or algorithm instance itself. For example "raster info", or ["raster", "info"] or "raster", "info".
- **arguments** (*dict*) -- Input arguments of the algorithm. For example {"format": "json", "input": "byte.tif"}

- **progress** (*callable*) -- Progress function whose arguments are a progress ratio, a string and a user data
- **kwargs** -- Instead of using the `arguments` parameter, it is possible to pass algorithm arguments directly as named parameters of gdal.Run(). If the named argument has dash characters in it, the corresponding parameter must replace them with an underscore character. For example `output_crs` as a a parameter of gdal.Run(), instead of `output-crs` which is the name to use on the command line.

Return type
a *osgeo.gdal.Algorithm* instance

If you do not need to access output value(s) of the algorithm, you can call `Run` directly: `gdal.Run(["raster", "convert"], input="in.tif", output="out.tif")`

The name of algorithm arguments are the long names as documented in the documentation page of each algorithm. They can also be obtained with *osgeo.gdal.Algorithm.GetArgNames()*.

```
>>> gdal.Algorithm("raster", "convert").GetArgNames()
['help', 'help-doc', 'version', 'json-usage', 'drivers', 'config', 'progress', 'output-
↪format', 'open-option', 'input-format', 'input', 'output', 'creation-option',
↪'overwrite', 'append']
```

For a command such as *gdal raster info* that returns a JSON output, you can get the return value of *osgeo.gdal.Run()* and call the *osgeo.gdal.Algorithm.Output()* method.

```
alg = gdal.Run("raster", "info", input="byte.tif")
info_as_dict = alg.Output()
```

If the return value is a dataset, *osgeo.gdal.Run()* can be used within a context manager, in which case *osgeo.gdal.Algorithm.Finalize()* will be called at the exit of the context manager.

```
with gdal.Run("raster reproject", input=src_ds, output_format="MEM",
              dst_crs="EPSG:4326") as alg:
    values = alg.Output().ReadAsArray()
```

When not using a context manager, *osgeo.gdal.Algorithm.Finalize()* can be called directly to release the algorithm's references to the output dataset and free associated resources. If a variable still references the dataset, it will not be closed and may still be accessed:

```
>>> alg = gdal.Run("raster reproject", input=src_ds, output_format="MEM",
               dst_crs="EPSG:4326")
>>> out_ds = alg.Output()
>>> print(out_ds.GetRefCount())
2
>>> alg.Finalize()
>>> print(out_ds.GetRefCount())
1
>>> alg.Output().ReadAsArray() # no longer available from the algorithm
AttributeError: 'NoneType' object has no attribute 'ReadAsMaskedArray'
>>> out_ds.ReadAsArray() # still accessible via the existing reference
array([[-9999., -9999., -9999., ...,
```

gdal.alg module

Added in version 3.12.

The gdal.alg module is also available with submodules and functions reflecting the hierarchy of algorithms. Functions return an *osgeo.gdal.Algorithm* instance similarly to *osgeo.gdal.Run()*

The above example running "gdal raster convert" can also be rewritten as:

```
>>> from osgeo import gdal
>>> gdal.alg.raster.convert(input="in.tif", output="out.tif")
```

The documentation of the function lists argument names, types and semantics:

```
>>> from osgeo import gdal
>>> help(gdal.alg.raster.convert)

    Help on function convert:

    convert(input: Union[List[osgeo.gdal.Dataset], List[str], List[os.PathLike[str]]],␣
↪output: Union[osgeo.gdal.Dataset, str, os.PathLike[str]], input_format:␣
↪Union[List[str], str, NoneType] = None, open_option: Union[List[str], str, NoneType] =␣
↪None, output_format: Optional[str] = None, creation_option: Union[List[str], str,␣
↪NoneType] = None, overwrite: Optional[bool] = None, append: Optional[bool] = None)
        Convert a raster dataset.

        Consult https://gdal.org/programs/gdal_raster_convert.html for more details.

        Parameters
        ----------
        input: Union[List[gdal.Dataset], List[str], List[os.PathLike[str]]]
            Input raster datasets
        output: Union[gdal.Dataset, str, os.PathLike[str]]
            Output raster dataset
        input_format: Optional[Union[List[str], str]]=None
            Input formats
        open_option: Optional[Union[List[str], dict, str]]=None
            Open options
        output_format: Optional[str]=None
            Output format ("GDALG" allowed)
        creation_option: Optional[Union[List[str], dict, str]]=None
            Creation option
        overwrite: Optional[bool]=None
            Whether overwriting existing output is allowed
        append: Optional[bool]=None
            Append as a subdataset to existing output


        Output parameters
        -----------------
        output: gdal.ArgDatasetValue
            Output raster dataset
```

Raster commands examples

Example 1: Getting information on a raster dataset as JSON

```
from osgeo import gdal

gdal.UseExceptions()
info_as_dict = gdal.alg.raster.info(input="byte.tif").Output()
```

Example 2: Converting a georeferenced netCDF file to cloud-optimized GeoTIFF

```
from osgeo import gdal

gdal.UseExceptions()
gdal.alg.raster.convert(input="in.tif", output="out.tif", output_format="COG",␣
↪overwrite=True)
```

or

```
from osgeo import gdal

gdal.UseExceptions()
gdal.Run("raster", "convert", input="in.tif", output="out.tif",
         output_format="COG", overwrite=True)
```

or

```
from osgeo import gdal

gdal.UseExceptions()
gdal.Run(["raster", "convert"], {"input": "in.tif", "output": "out.tif", "output-format
↪": "COG", "overwrite": True})
```

Example 3: Reprojecting a GeoTIFF file to a Deflate-compressed tiled GeoTIFF file

```
from osgeo import gdal

gdal.UseExceptions()
gdal.alg.raster.reproject(
          input="in.tif", output="out.tif",
          dst_crs="EPSG:4326",
          creation_options={ "TILED": "YES", "COMPRESS": "DEFLATE"})
```

Example 4: Reprojecting a (possibly in-memory) dataset to a in-memory dataset

```
from osgeo import gdal

gdal.UseExceptions()
with gdal.alg.raster.reproject(input=src_ds, output_format="MEM",
               dst_crs="EPSG:4326") as alg:
    values = alg.Output().ReadAsArray()
```

Vector commands examples

Example 5: Getting information on a vector dataset as JSON

```
from osgeo import gdal

gdal.UseExceptions()
info_as_dict = gdal.alg.vector.info(input="poly.gpkg").Output()
```

Example 6: Converting a shapefile to a GeoPackage

```
from osgeo import gdal

gdal.UseExceptions()
gdal.alg.vector.convert(input="in.shp", output="out.gpkg", overwrite=True)
```

Pipeline examples

Example 7: Perform raster reprojection and gets the result as a streamed dataset.

```
from osgeo import gdal

gdal.UseExceptions()
with gdal.alg.pipeline(pipeline="read byte.tif ! reproject --output-crs EPSG:4326 --
↪resampling cubic") as alg:
    ds = alg.Output()
    # do something with the dataset
```

5.1.1.8 .gdalg files to replay serialized `gdal` commands

A subset of subcommands of **`gdal`** support generating on-the-fly / streamed raster of vector datasets. They are typically saved to a "real" output dataset, but it is also possible to save them in a JSON file of extension `.gdalg.json`, that can be read by the *GDALG: GDAL Streamed Algorithm* driver or *GDALG: GDAL Streamed Algorithm* driver to apply the saved processing on-the-fly and use the resulting dataset as a regular input dataset for *`GDALOpenEx()`*, or anywhere in GDAL API or command line utilities where a raster input dataset is expected. GDALG files are conceptually close to *VRT (Virtual) files*, although the implementation is substantially different.

> **Note**
>
> GDALG is the contraction of GDAL and ALGorithm.

In-process stream execution

For algorithms that support GDALG output, it is also possible to use the `stream` output format to indicate that a raster or vector streamed dataset must be returned.

For example the following snippet, runs that "gdal vector set-type" algorithm on a source dataset and iterates over features from the returned streamed dataset.

```
from osgeo import gdal
gdal.UseExceptions()
```

(continues on next page)

(continued from previous page)

```
alg = gdal.GetGlobalAlgorithmRegistry()["vector"]["geom"]["set-type"]
alg["input"] = src_ds
alg["output"] = ""
alg["output-format"] = "stream"
alg["geometry-type"] = "LINESTRING Z"

alg.Run()

out_ds = alg["output"].GetDataset()
out_lyr = out_ds.GetLayer(0)
for f in our_lyr:
    f.DumpReadable()
```

Examples

Example 1: Serialize the command of a reprojection of a GeoPackage file in a GDALG file, and later read it

```
    $ gdal vector pipeline ! read in.gpkg ! reproject --output-crs=EPSG:32632 ! write in_
↪epsg_32632.gdalg.json --overwrite
    $ gdal vector info in_epsg_32632.gdalg.json

The content of :file:`in_epsg_32632.gdalg.json` is:

.. code-block:: json

    {
        "type": "gdal_streamed_alg",
        "command_line": "gdal vector pipeline ! read in.gpkg ! reproject --output-
↪crs=EPSG:32632"
    }
```

5.1.2 Commands working with raster or vector inputs

5.1.2.1 gdal info

Added in version 3.11.

Acts as a shortcut for *gdal raster info* or *gdal vector info* depending on the nature of the specified dataset.

Synopsis

```
Usage: gdal info [OPTIONS] <INPUT>

Return information on a dataset (shortcut for 'gdal raster info' or 'gdal vector info').

Positional arguments:
  -i, --dataset, --input <INPUT>                        Input raster, vector or␣
↪multidimensional raster dataset [required]

Common Options:
```

(continues on next page)

(continued from previous page)

```
  -h, --help                                          Display help message and exit
  --json-usage                                        Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                              Configuration option [may be␣
↪repeated]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format. OUTPUT-
↪FORMAT=json|text

For all options, run 'gdal raster info --help' or 'gdal vector info --help'
```

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Getting information on the file utmsmall.tif (with text output)

```
$ gdal info utmsmall.tif
Driver: GTiff/GeoTIFF
Files: utmsmall.tif
Size is 100, 100
Coordinate Reference System:
  - name: NAD27 / UTM zone 11N
  - ID: EPSG:26711
  - type: Projected
  - projection type: Transverse Mercator
  - units: metre
Data axis to CRS axis mapping: 1,2
Origin = (440720.000000000000000,3751320.000000000000000)
Pixel Size = (60.000000000000000,-60.000000000000000)
Metadata:
  AREA_OR_POINT=Area
Image Structure Metadata:
  INTERLEAVE=BAND
Corner Coordinates:
Upper Left  (  440720.000, 3751320.000) (117d38'28.21"W, 33d54' 8.47"N)
Lower Left  (  440720.000, 3745320.000) (117d38'26.75"W, 33d50'53.66"N)
Upper Right (  446720.000, 3751320.000) (117d34'34.59"W, 33d54' 9.62"N)
Lower Right (  446720.000, 3745320.000) (117d34'33.28"W, 33d50'54.82"N)
Center      (  443720.000, 3748320.000) (117d36'30.71"W, 33d52'31.66"N)
Band 1 Block=100x81 Type=Byte, ColorInterp=Gray
```

Example 2: Getting information on the file poly.gpkg (with JSON output), listing all features

```
$ gdal vector info --format=JSON --features poly.gpkg
{
  "description":"poly.gpkg",
```

(continues on next page)

(continued from previous page)


```
  "driverShortName":"GPKG",
  "driverLongName":"GeoPackage",
  "layers":[
    {
      "name":"poly",
      "metadata":{
        "":{
          "DBF_DATE_LAST_UPDATE":"2018-08-02"
        }
      },
      "geometryFields":[
        {
          "name":"geom",
          "type":"Polygon",
          "nullable":true,
          "extent":[
            478315.53125,
            4762880.5,
            481645.3125,
            4765610.5
          ],
          "coordinateSystem":{
            "wkt":"PROJCRS[\"OSGB36 / British National Grid\",\n    BASEGEOGCRS[\"OSGB36\
↪",\n        DATUM[\"Ordnance Survey of Great Britain 1936\",\n            ELLIPSOID[\
↪"Airy 1830\",6377563.396,299.3249646,\n                LENGTHUNIT[\"metre\",1]]],\n    ␣
↪    PRIMEM[\"Greenwich\",0,\n            ANGLEUNIT[\"degree\",0.0174532925199433]],\n ␣
↪      ID[\"EPSG\",4277]],\n    CONVERSION[\"British National Grid\",\n        METHOD[\
↪"Transverse Mercator\",\n            ID[\"EPSG\",9807]],\n        PARAMETER[\"Latitude␣
↪of natural origin\",49,\n            ANGLEUNIT[\"degree\",0.0174532925199433],\n      ␣
↪     ID[\"EPSG\",8801]],\n        PARAMETER[\"Longitude of natural origin\",-2,\n     ␣
↪      ANGLEUNIT[\"degree\",0.0174532925199433],\n            ID[\"EPSG\",8802]],\n    ␣
↪   PARAMETER[\"Scale factor at natural origin\",0.9996012717,\n            SCALEUNIT[\
↪"unity\",1],\n            ID[\"EPSG\",8805]],\n        PARAMETER[\"False easting\",
↪400000,\n            LENGTHUNIT[\"metre\",1],\n            ID[\"EPSG\",8806]],\n      ␣
↪ PARAMETER[\"False northing\",-100000,\n            LENGTHUNIT[\"metre\",1],\n        ␣
↪   ID[\"EPSG\",8807]]],\n    CS[Cartesian,2],\n        AXIS[\"(E)\",east,\n           ␣
↪ORDER[1],\n            LENGTHUNIT[\"metre\",1]],\n        AXIS[\"(N)\",north,\n       ␣
↪    ORDER[2],\n            LENGTHUNIT[\"metre\",1]],\n    USAGE[\n        SCOPE[\
↪"Engineering survey, topographic mapping.\"],\n        AREA[\"United Kingdom (UK) -␣
↪offshore to boundary of UKCS within 49°45'N to 61°N and 9°W to 2°E; onshore Great␣
↪Britain (England, Wales and Scotland). Isle of Man onshore.\"],\n        BBOX[49.75,-9.
↪01,61.01,2.01]],\n    ID[\"EPSG\",27700]]",
            "projjson":{
              "$schema":"https://proj.org/schemas/v0.7/projjson.schema.json",
              "type":"ProjectedCRS",
              "name":"OSGB36 / British National Grid",
              "base_crs":{
                "name":"OSGB36",
                "datum":{
                  "type":"GeodeticReferenceFrame",
                  "name":"Ordnance Survey of Great Britain 1936",
                  "ellipsoid":{
```


(continues on next page)

(continued from previous page)

```
                "name":"Airy 1830",
                "semi_major_axis":6377563.396,
                "inverse_flattening":299.3249646
              }
            },
            "coordinate_system":{
              "subtype":"ellipsoidal",
              "axis":[
                {
                  "name":"Geodetic latitude",
                  "abbreviation":"Lat",
                  "direction":"north",
                  "unit":"degree"
                },
                {
                  "name":"Geodetic longitude",
                  "abbreviation":"Lon",
                  "direction":"east",
                  "unit":"degree"
                }
              ]
            },
            "id":{
              "authority":"EPSG",
              "code":4277
            }
          },
          "conversion":{
            "name":"British National Grid",
            "method":{
              "name":"Transverse Mercator",
              "id":{
                "authority":"EPSG",
                "code":9807
              }
            },
            "parameters":[
              {
                "name":"Latitude of natural origin",
                "value":49,
                "unit":"degree",
                "id":{
                  "authority":"EPSG",
                  "code":8801
                }
              },
              {
                "name":"Longitude of natural origin",
                "value":-2,
                "unit":"degree",
                "id":{
                  "authority":"EPSG",
```

(continues on next page)

(continued from previous page)

```
                    "code":8802
                  }
                },
                {
                  "name":"Scale factor at natural origin",
                  "value":0.9996012717,
                  "unit":"unity",
                  "id":{
                    "authority":"EPSG",
                    "code":8805
                  }
                },
                {
                  "name":"False easting",
                  "value":400000,
                  "unit":"metre",
                  "id":{
                    "authority":"EPSG",
                    "code":8806
                  }
                },
                {
                  "name":"False northing",
                  "value":-100000,
                  "unit":"metre",
                  "id":{
                    "authority":"EPSG",
                    "code":8807
                  }
                }
              ]
            },
            "coordinate_system":{
              "subtype":"Cartesian",
              "axis":[
                {
                  "name":"Easting",
                  "abbreviation":"E",
                  "direction":"east",
                  "unit":"metre"
                },
                {
                  "name":"Northing",
                  "abbreviation":"N",
                  "direction":"north",
                  "unit":"metre"
                }
              ]
            },
            "scope":"Engineering survey, topographic mapping.",
            "area":"United Kingdom (UK) - offshore to boundary of UKCS within 49°45'N␣
↪to 61°N and 9°W to 2°E; onshore Great Britain (England, Wales and Scotland). Isle of␣
```

(continues on next page)

(continued from previous page)

```
↪Man onshore.",
              "bbox":{
                "south_latitude":49.75,
                "west_longitude":-9.01,
                "north_latitude":61.01,
                "east_longitude":2.01
              },
              "id":{
                "authority":"EPSG",
                "code":27700
              }
            },
            "dataAxisToSRSAxisMapping":[
              1,
              2
            ]
          }
        }
      ],
      "featureCount":10,
      "fidColumnName":"fid",
      "fields":[
        {
          "name":"AREA",
          "type":"Real",
          "nullable":true,
          "uniqueConstraint":false
        },
        {
          "name":"EAS_ID",
          "type":"Integer64",
          "nullable":true,
          "uniqueConstraint":false
        },
        {
          "name":"PRFEDEA",
          "type":"String",
          "width":16,
          "nullable":true,
          "uniqueConstraint":false
        }
      ],
      "features":[
        {
          "type":"Feature",
          "properties":{
            "AREA":215229.266,
            "EAS_ID":168,
            "PRFEDEA":"35043411"
          },
          "fid":1,
          "geometry":{
```

(continues on next page)

(continued from previous page)

```
            "type":"Polygon",
            "coordinates":[
              [
                [
                  479819.84375,
                  4765180.5
                ],
                [
                  479690.1875,
                  4765259.5
                ],
                [
                  479647.0,
                  4765369.5
                ],
                [
                  479730.375,
                  4765400.5
                ],
                [
                  480039.03125,
                  4765539.5
                ],
                [
                  480035.34375,
                  4765558.5
                ],
                [
                  480159.78125,
                  4765610.5
                ],
                [
                  480202.28125,
                  4765482.0
                ],
                [
                  480365.0,
                  4765015.5
                ],
                [
                  480389.6875,
                  4764950.0
                ],
                [
                  480133.96875,
                  4764856.5
                ],
                [
                  480080.28125,
                  4764979.5
                ],
                [
```

(continues on next page)

(continued from previous page)

```
                480082.96875,
                4765049.5
              ],
              [
                480088.8125,
                4765139.5
              ],
              [
                480059.90625,
                4765239.5
              ],
              [
                480019.71875,
                4765319.5
              ],
              [
                479980.21875,
                4765409.5
              ],
              [
                479909.875,
                4765370.0
              ],
              [
                479859.875,
                4765270.0
              ],
              [
                479819.84375,
                4765180.5
              ]
            ]
          ]
        }
      },
      {
        "type":"Feature",
        "properties":{
          "AREA":247328.172,
          "EAS_ID":179,
          "PRFEDEA":"35043423"
        },
        "fid":2,
        "geometry":{
          "type":"Polygon",
          "coordinates":[
            [
              [
                480035.34375,
                4765558.5
              ],
              [
```

(continues on next page)

(continued from previous page)

```
                480039.03125,
                4765539.5
              ],
              [
                479730.375,
                4765400.5
              ],
              [
                479647.0,
                4765369.5
              ],
              [
                479690.1875,
                4765259.5
              ],
              [
                479819.84375,
                4765180.5
              ],
              [
                479779.84375,
                4765109.5
              ],
              [
                479681.78125,
                4764940.0
              ],
              [
                479468.0,
                4764942.5
              ],
              [
                479411.4375,
                4764940.5
              ],
              [
                479353.0,
                4764939.5
              ],
              [
                479208.65625,
                4764882.5
              ],
              [
                479196.8125,
                4764879.0
              ],
              [
                479123.28125,
                4765015.0
              ],
              [
```

(continues on next page)

(continued from previous page)

```
                479046.53125,
                4765117.0
              ],
              [
                479029.71875,
                4765110.5
              ],
              [
                479014.9375,
                4765147.5
              ],
              [
                479149.9375,
                4765200.5
              ],
              [
                479639.625,
                4765399.5
              ],
              [
                480035.34375,
                4765558.5
              ]
            ]
          ]
        }
      },
      {
        "type":"Feature",
        "properties":{
          "AREA":261752.781,
          "EAS_ID":171,
          "PRFEDEA":"35043414"
        },
        "fid":3,
        "geometry":{
          "type":"Polygon",
          "coordinates":[
            [
              [
                479819.84375,
                4765180.5
              ],
              [
                479859.875,
                4765270.0
              ],
              [
                479909.875,
                4765370.0
              ],
              [
```

(continues on next page)

(continued from previous page)

```
                479980.21875,
                4765409.5
              ],
              [
                480019.71875,
                4765319.5
              ],
              [
                480059.90625,
                4765239.5
              ],
              [
                480088.8125,
                4765139.5
              ],
              [
                480082.96875,
                4765049.5
              ],
              [
                480000.28125,
                4765043.0
              ],
              [
                479934.96875,
                4765020.0
              ],
              [
                479895.125,
                4765000.0
              ],
              [
                479734.375,
                4764865.0
              ],
              [
                479680.28125,
                4764852.0
              ],
              [
                479644.78125,
                4764827.5
              ],
              [
                479637.875,
                4764803.0
              ],
              [
                479617.21875,
                4764760.0
              ],
              [
```

(continues on next page)

(continued from previous page)

```
                479587.28125,
                4764718.0
              ],
              [
                479548.03125,
                4764693.5
              ],
              [
                479504.90625,
                4764609.5
              ],
              [
                479239.8125,
                4764505.0
              ],
              [
                479117.8125,
                4764847.0
              ],
              [
                479196.8125,
                4764879.0
              ],
              [
                479208.65625,
                4764882.5
              ],
              [
                479353.0,
                4764939.5
              ],
              [
                479411.4375,
                4764940.5
              ],
              [
                479468.0,
                4764942.5
              ],
              [
                479681.78125,
                4764940.0
              ],
              [
                479779.84375,
                4765109.5
              ],
              [
                479819.84375,
                4765180.5
              ]
            ]
```

(continues on next page)

(continued from previous page)


```
            ]
          }
        },
        {
          "type":"Feature",
          "properties":{
            "AREA":547597.188,
            "EAS_ID":173,
            "PRFEDEA":"35043416"
          },
          "fid":4,
          "geometry":{
            "type":"Polygon",
            "coordinates":[
              [
                [
                  479014.9375,
                  4765147.5
                ],
                [
                  479029.71875,
                  4765110.5
                ],
                [
                  479117.8125,
                  4764847.0
                ],
                [
                  479239.8125,
                  4764505.0
                ],
                [
                  479305.875,
                  4764361.0
                ],
                [
                  479256.03125,
                  4764314.5
                ],
                [
                  479220.90625,
                  4764212.5
                ],
                [
                  479114.5,
                  4764174.0
                ],
                [
                  479018.28125,
                  4764418.5
                ],
                [
```


(continues on next page)

(continued from previous page)

```
                478896.9375,
                4764371.0
              ],
              [
                478748.8125,
                4764308.5
              ],
              [
                478503.03125,
                4764218.0
              ],
              [
                478461.75,
                4764337.5
              ],
              [
                478443.9375,
                4764400.5
              ],
              [
                478447.8125,
                4764454.0
              ],
              [
                478448.6875,
                4764531.5
              ],
              [
                478502.1875,
                4764541.5
              ],
              [
                478683.0,
                4764730.5
              ],
              [
                478621.03125,
                4764788.5
              ],
              [
                478597.34375,
                4764766.5
              ],
              [
                478532.5,
                4764695.5
              ],
              [
                478460.125,
                4764615.0
              ],
              [
```

(continues on next page)

(continued from previous page)

```
                478408.0625,
                4764654.0
              ],
              [
                478315.53125,
                4764876.0
              ],
              [
                478889.25,
                4765100.0
              ],
              [
                479014.9375,
                4765147.5
              ]
            ]
          ]
        }
      },
      {
        "type":"Feature",
        "properties":{
          "AREA":15775.758,
          "EAS_ID":172,
          "PRFEDEA":"35043415"
        },
        "fid":5,
        "geometry":{
          "type":"Polygon",
          "coordinates":[
            [
              [
                479029.71875,
                4765110.5
              ],
              [
                479046.53125,
                4765117.0
              ],
              [
                479123.28125,
                4765015.0
              ],
              [
                479196.8125,
                4764879.0
              ],
              [
                479117.8125,
                4764847.0
              ],
              [
```

(continues on next page)

(continued from previous page)

```
                479029.71875,
                4765110.5
              ]
            ]
          ]
        }
      },
      {
        "type":"Feature",
        "properties":{
          "AREA":101429.977,
          "EAS_ID":169,
          "PRFEDEA":"35043412"
        },
        "fid":6,
        "geometry":{
          "type":"Polygon",
          "coordinates":[
            [
              [
                480082.96875,
                4765049.5
              ],
              [
                480080.28125,
                4764979.5
              ],
              [
                480133.96875,
                4764856.5
              ],
              [
                479968.46875,
                4764788.0
              ],
              [
                479750.6875,
                4764702.0
              ],
              [
                479735.90625,
                4764752.0
              ],
              [
                479640.09375,
                4764721.0
              ],
              [
                479658.59375,
                4764670.0
              ],
              [
```

(continues on next page)

(continued from previous page)

```
                479504.90625,
                4764609.5
              ],
              [
                479548.03125,
                4764693.5
              ],
              [
                479587.28125,
                4764718.0
              ],
              [
                479617.21875,
                4764760.0
              ],
              [
                479637.875,
                4764803.0
              ],
              [
                479644.78125,
                4764827.5
              ],
              [
                479680.28125,
                4764852.0
              ],
              [
                479734.375,
                4764865.0
              ],
              [
                479895.125,
                4765000.0
              ],
              [
                479934.96875,
                4765020.0
              ],
              [
                480000.28125,
                4765043.0
              ],
              [
                480082.96875,
                4765049.5
              ]
            ]
          ]
        }
      },
      {
```

(continues on next page)

(continued from previous page)

```
        "type":"Feature",
        "properties":{
          "AREA":268597.625,
          "EAS_ID":166,
          "PRFEDEA":"35043409"
        },
        "fid":7,
        "geometry":{
          "type":"Polygon",
          "coordinates":[
            [
              [
                480389.6875,
                4764950.0
              ],
              [
                480537.15625,
                4765014.0
              ],
              [
                480567.96875,
                4764918.0
              ],
              [
                480605.0,
                4764835.0
              ],
              [
                480701.0625,
                4764738.0
              ],
              [
                480710.25,
                4764690.5
              ],
              [
                480588.59375,
                4764740.5
              ],
              [
                480540.71875,
                4764741.0
              ],
              [
                480515.125,
                4764695.0
              ],
              [
                480731.65625,
                4764561.5
              ],
              [
```

(continues on next page)

(continued from previous page)

```
                480692.1875,
                4764453.5
              ],
              [
                480677.84375,
                4764439.0
              ],
              [
                480655.34375,
                4764397.5
              ],
              [
                480584.375,
                4764353.0
              ],
              [
                480500.40625,
                4764326.5
              ],
              [
                480358.53125,
                4764277.0
              ],
              [
                480192.3125,
                4764183.0
              ],
              [
                480157.125,
                4764266.5
              ],
              [
                480234.3125,
                4764304.0
              ],
              [
                480289.125,
                4764348.5
              ],
              [
                480316.0,
                4764395.0
              ],
              [
                480343.5625,
                4764477.0
              ],
              [
                480343.71875,
                4764532.5
              ],
              [
```

(continues on next page)

(continued from previous page)

```
                480258.03125,
                4764767.0
              ],
              [
                480177.15625,
                4764742.0
              ],
              [
                480093.75,
                4764703.0
              ],
              [
                480011.0,
                4764674.5
              ],
              [
                479985.0625,
                4764732.0
              ],
              [
                479968.46875,
                4764788.0
              ],
              [
                480133.96875,
                4764856.5
              ],
              [
                480389.6875,
                4764950.0
              ]
            ]
          ]
        }
      },
      {
        "type":"Feature",
        "properties":{
          "AREA":1634833.375,
          "EAS_ID":158,
          "PRFEDEA":"35043369"
        },
        "fid":8,
        "geometry":{
          "type":"Polygon",
          "coordinates":[
            [
              [
                480701.0625,
                4764738.0
              ],
              [
```

(continues on next page)

(continued from previous page)

```
                480761.46875,
                4764778.0
              ],
              [
                480824.96875,
                4764820.0
              ],
              [
                480922.03125,
                4764850.5
              ],
              [
                480930.71875,
                4764852.0
              ],
              [
                480984.25,
                4764875.0
              ],
              [
                481088.1875,
                4764936.0
              ],
              [
                481136.84375,
                4764994.5
              ],
              [
                481281.3125,
                4764876.5
              ],
              [
                481291.09375,
                4764810.0
              ],
              [
                481465.90625,
                4764872.5
              ],
              [
                481457.375,
                4764937.0
              ],
              [
                481509.65625,
                4764967.0
              ],
              [
                481538.90625,
                4764982.5
              ],
              [
```

(continues on next page)

(continued from previous page)

```
                481575.0,
                4764999.5
              ],
              [
                481602.125,
                4764915.5
              ],
              [
                481629.84375,
                4764829.5
              ],
              [
                481645.3125,
                4764797.5
              ],
              [
                481635.96875,
                4764795.5
              ],
              [
                481235.3125,
                4764650.0
              ],
              [
                481209.8125,
                4764633.5
              ],
              [
                481199.21875,
                4764623.5
              ],
              [
                481185.5,
                4764607.0
              ],
              [
                481159.9375,
                4764580.0
              ],
              [
                481140.46875,
                4764510.5
              ],
              [
                481141.625,
                4764480.5
              ],
              [
                481199.84375,
                4764180.0
              ],
              [
```

(continues on next page)

(continued from previous page)

```
                481143.4375,
                4764010.5
              ],
              [
                481130.3125,
                4763979.5
              ],
              [
                481039.9375,
                4763889.5
              ],
              [
                480882.6875,
                4763670.0
              ],
              [
                480826.0625,
                4763650.5
              ],
              [
                480745.1875,
                4763628.5
              ],
              [
                480654.4375,
                4763627.5
              ],
              [
                480599.8125,
                4763660.0
              ],
              [
                480281.9375,
                4763576.5
              ],
              [
                480221.5,
                4763533.5
              ],
              [
                480199.6875,
                4763509.0
              ],
              [
                480195.09375,
                4763430.0
              ],
              [
                480273.6875,
                4763305.5
              ],
              [
```

(continues on next page)

(continued from previous page)

```
                480309.6875,
                4763063.5
              ],
              [
                480201.84375,
                4762962.5
              ],
              [
                479855.3125,
                4762880.5
              ],
              [
                479848.53125,
                4762897.0
              ],
              [
                479728.875,
                4763217.5
              ],
              [
                479492.6875,
                4763850.0
              ],
              [
                479550.0625,
                4763919.5
              ],
              [
                480120.21875,
                4764188.5
              ],
              [
                480192.3125,
                4764183.0
              ],
              [
                480358.53125,
                4764277.0
              ],
              [
                480500.40625,
                4764326.5
              ],
              [
                480584.375,
                4764353.0
              ],
              [
                480655.34375,
                4764397.5
              ],
              [
```

(continues on next page)

(continued from previous page)

```
                480677.84375,
                4764439.0
              ],
              [
                480692.1875,
                4764453.5
              ],
              [
                480731.65625,
                4764561.5
              ],
              [
                480515.125,
                4764695.0
              ],
              [
                480540.71875,
                4764741.0
              ],
              [
                480588.59375,
                4764740.5
              ],
              [
                480710.25,
                4764690.5
              ],
              [
                480701.0625,
                4764738.0
              ]
            ]
          ]
        }
      },
      {
        "type":"Feature",
        "properties":{
          "AREA":-596610.313,
          "EAS_ID":165,
          "PRFEDEA":"35043408"
        },
        "fid":9,
        "geometry":{
          "type":"Polygon",
          "coordinates":[
            [
              [
                479750.6875,
                4764702.0
              ],
              [
```

(continues on next page)

(continued from previous page)

```
                479968.46875,
                4764788.0
              ],
              [
                479985.0625,
                4764732.0
              ],
              [
                480011.0,
                4764674.5
              ],
              [
                480093.75,
                4764703.0
              ],
              [
                480177.15625,
                4764742.0
              ],
              [
                480258.03125,
                4764767.0
              ],
              [
                480343.71875,
                4764532.5
              ],
              [
                480343.5625,
                4764477.0
              ],
              [
                480316.0,
                4764395.0
              ],
              [
                480289.125,
                4764348.5
              ],
              [
                480234.3125,
                4764304.0
              ],
              [
                480157.125,
                4764266.5
              ],
              [
                480192.3125,
                4764183.0
              ],
              [
```

(continues on next page)

(continued from previous page)

```
                480120.21875,
                4764188.5
              ],
              [
                479550.0625,
                4763919.5
              ],
              [
                479492.6875,
                4763850.0
              ],
              [
                479487.75,
                4763864.5
              ],
              [
                479442.75,
                4763990.0
              ],
              [
                479436.0,
                4764023.0
              ],
              [
                479398.9375,
                4764100.0
              ],
              [
                479349.625,
                4764230.0
              ],
              [
                479305.875,
                4764361.0
              ],
              [
                479239.8125,
                4764505.0
              ],
              [
                479504.90625,
                4764609.5
              ],
              [
                479658.59375,
                4764670.0
              ],
              [
                479750.6875,
                4764702.0
              ]
            ]
```

(continues on next page)

(continued from previous page)


```
            ]
          }
        },
        {
          "type":"Feature",
          "properties":{
            "AREA":5268.813,
            "EAS_ID":170,
            "PRFEDEA":"35043413"
          },
          "fid":10,
          "geometry":{
            "type":"Polygon",
            "coordinates":[
              [
                [
                  479750.6875,
                  4764702.0
                ],
                [
                  479658.59375,
                  4764670.0
                ],
                [
                  479640.09375,
                  4764721.0
                ],
                [
                  479735.90625,
                  4764752.0
                ],
                [
                  479750.6875,
                  4764702.0
                ]
              ]
            ]
          }
        }
      ]
    }
  ],
  "metadata":{
  },
  "domains":{
  },
  "relationships":{
  }
}
```

Example 3: Getting the list of all drivers (with JSON output)

```
$ gdal --drivers
```

5.1.2.2 gdal convert

Added in version 3.11.

Acts as a shortcut for *gdal raster convert* or *gdal vector convert* depending on the nature of the specified dataset.

Synopsis

```
Usage: gdal convert [OPTIONS] <INPUT> <OUTPUT>

Convert a dataset (shortcut for 'gdal raster convert' or 'gdal vector convert').

Positional arguments:
  -i, --input <INPUT>                                  Input raster, vector or␣
↪multidimensional raster dataset [required]
  -o, --output <OUTPUT>                                Output raster, vector or␣
↪multidimensional raster dataset (created by algorithm) [required]

Common Options:
  -h, --help                                           Display help message and exit
  --json-usage                                         Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                               Configuration option [may be␣
↪repeated]
  -q, --quiet                                          Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format

For all options, run 'gdal raster convert --help' or 'gdal vector convert --help'
```

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Converting file utm.tif to GeoPackage raster

```
$ gdal convert utm.tif utm.gpkg
```

5.1.2.3 gdal pipeline

Added in version 3.12.

Description

gdal pipeline execute a pipeline, taking a raster or input dataset, execute steps and finally writing a raster or vector dataset.

Most steps proceed in on-demand evaluation of raster blocks or features, unless otherwise stated in their documentation, without "materializing" the resulting dataset of the operation of each step. It may be desirable sometimes for performance purposes to proceed to materializing an intermediate dataset to disk using *gdal raster materialize* or *gdal vector materialize*.

Synopsis

```
Usage: gdal pipeline [OPTIONS] <PIPELINE>

Process a dataset applying several steps.

Positional arguments:

Common Options:
  -h, --help                        Display help message and exit
  --json-usage                      Display usage as JSON document and exit
  --config <KEY>=<VALUE>            Configuration option [may be repeated]
  -q, --quiet                       Quiet mode (no progress bar or warning message) [not␣
↪available in pipelines]

Advanced Options:
  --<user-provided-option>=<value>  Argument provided by user

<PIPELINE> is of the form: read|calc|concat|create|mosaic|stack [READ-OPTIONS] ( ! <STEP-
↪NAME> [STEP-OPTIONS] )* ! write!info!tile [WRITE-OPTIONS]
```

A pipeline chains several steps, separated with the `!` (exclamation mark) character. Including a `!` between `gdal pipeline` and the first step is optional. The first step must be `read`, `calc`, `concat`, `mosaic` or `stack`, and the last one `export-schema`, `info`, `tile` or `write`. Each step has its own positional or non-positional arguments. Apart from `read`, `calc`, `concat`, `mosaic`, `stack`, `info`, `tile`, `partition` and `write`, all other steps can potentially be used several times in a pipeline.

Example 1: Compute the footprint of a raster and apply a buffer on the footprint

```
$ gdal pipeline ! read in.tif ! footprint ! buffer 20 ! write out.gpkg --overwrite
```

Steps

For steps that have both *raster* data type as input and output, consult *gdal raster pipeline*. For steps that have both *vector* data type as input and output, consult *gdal vector pipeline*.

The table below lists steps that convert between raster and vector data.

Step	Direction
contour	Raster → Vector
footprint	Raster → Vector
pixel-info	Raster → Vector
polygonize	Raster → Vector
grid	Vector → Raster
rasterize	Vector → Raster
tee	Vector → Raster

contour

```
* contour [OPTIONS]
-------------------

Creates a vector contour from a raster elevation model (DEM).

Options:
  --output-layer <OUTPUT-LAYER>              Output layer name
  -b, --band <BAND>                          Input band (1-based index) (default: 1)
  --elevation-name <ELEVATION-NAME>          Name of the elevation field
  --min-name <MIN-NAME>                      Name of the minimum elevation field
  --max-name <MAX-NAME>                      Name of the maximum elevation field
  --3d                                       Force production of 3D vectors instead of 2D
  --input-nodata <INPUT-NODATA>              Input pixel value to treat as 'nodata'
  --interval <INTERVAL>                      Elevation interval between contours
                                             Mutually exclusive with --levels, --exp-base
  --levels <LEVELS>                          List of contour levels [may be repeated]
                                             Mutually exclusive with --interval, --exp-
↪base
  -e, --exp-base <EXP-BASE>                  Base for exponential contour level␣
↪generation
                                             Mutually exclusive with --interval, --levels
  --off, --offset <OFFSET>                   Offset to apply to contour levels
  -p, --polygonize                           Create polygons instead of lines
  --group-transactions <GROUP-TRANSACTIONS>  Group n features per transaction (default␣
↪100 000)
```

Details for options can be found in *gdal raster contour*.

footprint

```
* footprint [OPTIONS]
---------------------

Compute the footprint of a raster dataset.

Options:
  --output-layer <OUTPUT-LAYER>              Output layer name (default: footprint)
  -b, --band <BAND>                          Input band(s) (1-based index) [may be␣
↪repeated]
  --combine-bands <COMBINE-BANDS>            Defines how the mask bands of the selected␣
```

(continues on next page)

(continued from previous page)

```
↪bands are combined to generate a single mask band, before being vectorized.. COMBINE-
↪BANDS=union|intersection (default: union)
  --overview <OVERVIEW>                      Which overview level of source file must be␣
↪used
                                             Mutually exclusive with --input-nodata
  --input-nodata <INPUT-NODATA>              Set nodata values for input bands. [1..␣
↪values]
                                             Mutually exclusive with --overview
  --coordinate-system <COORDINATE-SYSTEM>    Target coordinate system. COORDINATE-
↪SYSTEM=georeferenced|pixel
  --output-crs <OUTPUT-CRS>                  Output CRS
  --split-multipolygons                      Whether to split multipolygons as several␣
↪features each with one single polygon
  --convex-hull                              Whether to compute the convex hull of the␣
↪footprint
  --densify-distance <DENSIFY-DISTANCE>      Maximum distance between 2 consecutive␣
↪points of the output geometry.
  --simplify-tolerance <SIMPLIFY-TOLERANCE>  Tolerance used to merge consecutive points␣
↪of the output geometry.
  --min-ring-area <MIN-RING-AREA>            Minimum value for the area of a ring
  --max-points <MAX-POINTS>                  Maximum number of points of each output␣
↪geometry (default: 100)
  --location-field <LOCATION-FIELD>          Name of the field where the path of the␣
↪input dataset will be stored. (default: location)
                                             Mutually exclusive with --no-location-field
  --no-location-field                        Disable creating a field with the path of␣
↪the input dataset
                                             Mutually exclusive with --location-field
  --absolute-path                            Whether the path to the input dataset␣
↪should be stored as an absolute path
```

Details for options can be found in *gdal raster footprint*.

grid

```
* grid <COMMAND> [OPTIONS]
--------------------------

where <COMMAND> is one of:
  - average:                 Create a regular grid from scattered points using moving␣
↪average interpolation.
  - average-distance:        Create a regular grid from scattered points using the␣
↪average distance between the grid node (center of the search ellipse) and all of the␣
↪data points in the search ellipse.
  - average-distance-points: Create a regular grid from scattered points using the␣
↪average distance between the data points in the search ellipse.
  - count:                   Create a regular grid from scattered points using the␣
↪number of points in the search ellipse.
  - invdist:                 Create a regular grid from scattered points using weighted␣
↪inverse distance interpolation.
  - invdistnn:               Create a regular grid from scattered points using weighted␣
↪inverse distance interpolation nearest neighbour.
```

(continues on next page)

(continued from previous page)

```
  - linear:                  Create a regular grid from scattered points using linear/
↪barycentric interpolation.
  - maximum:                 Create a regular grid from scattered points using the␣
↪maximum value in the search ellipse.
  - minimum:                 Create a regular grid from scattered points using the␣
↪minimum value in the search ellipse.
  - nearest:                 Create a regular grid from scattered points using nearest␣
↪neighbor interpolation.
  - range:                   Create a regular grid from scattered points using the␣
↪difference between the minimum and maximum values in the search ellipse.
```

Details for options can be found in *gdal vector grid*.

pixel-info

```
* pixel-info [OPTIONS] <INPUT> [<POSITION-DATASET>]
--------------------------------------------------

Return information on a pixel of a raster dataset.

Positional arguments:
  -i, --dataset, --input <INPUT>                Input raster datasets [required]
  --position-dataset <POSITION-DATASET>         Vector dataset with coordinates
                                                Mutually exclusive with --position

Options:
  -l, --layer, --input-layer <INPUT-LAYER>      Input layer name
  -b, --band <BAND>                             Input band(s) (1-based index) [may be␣
↪repeated]
  --overview <OVERVIEW>                         Which overview level of source file must␣
↪be used
  -p, --pos, --position <column,line> or <X,Y>  Pixel position [may be repeated]
                                                Mutually exclusive with --position-
↪dataset
  --position-crs <POSITION-CRS>                 CRS of position (default is 'pixel' if
↪'position-dataset' not specified)
  -r, --resampling <RESAMPLING>                 Resampling algorithm for interpolation.␣
↪RESAMPLING=nearest|bilinear|cubic|cubicspline (default: nearest)
  --promote-pixel-value-to-z                    Whether to set the pixel value as Z␣
↪component of GeoJSON geometry
  --include-field <INCLUDE-FIELD>               Fields from coordinate dataset to␣
↪include in output (special values: ALL and NONE) (default: ALL) [may be repeated]
```

Details for options can be found in *gdal raster pixel-info*.

polygonize

```
* polygonize [OPTIONS]
----------------------

Create a polygon feature dataset from a raster band.
```

(continues on next page)

(continued from previous page)

```
Options:
  --output-layer <OUTPUT-LAYER>      Output layer name
  -b, --band <BAND>                  Input band (1-based index) (default: 1)
  --attribute-name <ATTRIBUTE-NAME>  Name of the field with the pixel value (default: DN)
  -c, --connect-diagonal-pixels      Consider diagonal pixels as connected
```

Details for options can be found in *gdal raster polygonize*.

rasterize

```
* rasterize [OPTIONS]
---------------------

Burns vector geometries into a raster.

Options:
  -b, --band <BAND>                                          The band(s) to burn values␣
↪into (1-based index) [may be repeated]
  --invert                                                   Invert the rasterization
  --all-touched                                              Enables the ALL_TOUCHED␣
↪rasterization option
  --burn <BURN>                                              Burn value [may be repeated]
  -a, --attribute-name <ATTRIBUTE-NAME>                      Attribute name
  --3d                                                       Indicates that a burn value␣
↪should be extracted from the Z values of the feature
  -l, --input-layer <INPUT-LAYER>                            Input layer name
                                                             Mutually exclusive with --sql
  --where <WHERE>                                            SQL where clause
  --sql <SQL>                                                SQL select statement
                                                             Mutually exclusive with --
↪input-layer
  --dialect <DIALECT>                                        SQL dialect
  --nodata <NODATA>                                          Assign a specified nodata␣
↪value to output bands
  --init <INIT>                                              Pre-initialize output bands␣
↪with specified value [may be repeated]
  --crs <CRS>                                                Override the projection for␣
↪the output file
  --transformer-option <NAME>=<VALUE>                        Set a transformer option␣
↪suitable to pass to GDALCreateGenImgProjTransformer2 [may be repeated]
  --extent <xmin>,<ymin>,<xmax>,<ymax>                       Set the target georeferenced␣
↪extent [4 values]
  --resolution <xres>,<yres>                                 Set the target resolution [2␣
↪values]
                                                             Mutually exclusive with --size
  --tap, --target-aligned-pixels                             (target aligned pixels) Align␣
↪the coordinates of the extent of the output file to the values of the resolution
                                                             Depends on --resolution
  --size <xsize>,<ysize>                                     Set the target size in pixels␣
↪and lines [2 values]
                                                             Mutually exclusive with --
```

(continues on next page)

(continued from previous page)

```
↪resolution
  --ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>  Output data type. OUTPUT-DATA-
↪TYPE=UInt8|Int8|UInt16|Int16|UInt32|Int32|UInt64|Int64|CInt16|CInt32|Float16|Float32|Float64|CFloat32
  --optimization <OPTIMIZATION>                            Force the algorithm used␣
↪(results are identical). OPTIMIZATION=AUTO|RASTER|VECTOR (default: AUTO)
```

Details for options can be found in *gdal vector rasterize*.

tee

```
* tee [OPTIONS] [<PIPELINE>...]
------------------------------

Pipes the input into the output stream and side nested pipelines.

Positional arguments:
  --tee-pipeline <PIPELINE>  Nested pipeline [1.. values]
```

Details for options can be found in *Output nested pipeline*.

GDALG output (on-the-fly / streamed dataset)

A pipeline can be serialized as a JSON file using the `GDALG` output format. The resulting file can then be opened as a dataset using the *GDALG: GDAL Streamed Algorithm* or *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

The `command_line` member of the JSON file should nominally be the whole command line without the final `write` step, and is what is generated by `gdal pipeline ! .... ! write out.gdalg.json`.

```
{
    "type": "gdal_streamed_alg",
    "command_line": "gdal pipeline ! read in.tif ! footprint ! buffer 20"
}
```

The final `write` step can be added but if so it must explicitly specify the `stream` output format and a non-significant output dataset name.

```
{
    "type": "gdal_streamed_alg",
    "command_line": "gdal pipeline ! read in.tif ! footprint ! buffer 20 ! write --
↪output-format=streamed streamed_dataset"
}
```

Substitutions

It is also possible to use **`gdal pipeline`** to use a pipeline already serialized in a `.gdalg.json` file, and customize its existing steps, typically changing an input filename, specifying an output filename, or adding/modifying arguments of steps.

The syntax is:

```
gdal pipeline <filename.gdalg.json> --<step-name>.<arg-name>=value
```

When specifying an existing argument of a step of a pipeline, the value from the pipeline is overridden by the one specified on the **gdal pipeline** command line.

Let's imagine we have a `raster_reproject.gdalg.json` with the following content:

```
{
    "type": "gdal_streamed_alg",
    "command_line": "gdal pipeline ! read in.tif ! reproject --output-crs=EPSG:4326 !␣
↪edit --metadata=CHANGES=reprojected"
}
```

It is possible to run it with the following command line, overriding the `input` argument of the `read` step, and implicitly adding a final `write` step with an `output` argument.

```
$ gdal pipeline raster_reproject.gdalg.json --read.input=other_input.tif --write.
↪output=out.tif
```

When there is no ambiguity, it is also possible to omit the step name, and just specify the argument name (if there is an ambiguity, **gdal pipeline** will emit an error, so this is safe to do):

```
$ gdal pipeline raster_reproject.gdalg.json --input=other_input.tif --output=out.tif --
↪co COMPRESS=LZW --overwrite
```

When a step appears several times in the pipeline, it must specified as `<step-name>[<idx>]`, where `<idx>` is a zero-based index.

For example, given:

```
{
    "type": "gdal_streamed_alg",
    "command_line": "gdal pipeline ! read in.tif ! edit --metadata=before=value !␣
↪reproject --output-crs=EPSG:4326 ! edit --metadata=CHANGES=reprojected"
}
```

the following command line may be used:

```
$ gdal pipeline raster_reproject.gdalg.json --edit[0].metadata=before=modified --
↪output=out.tif
```

Execution of pipelines and argument substitutions can also be done in Python with:

```
gdal.Run("pipeline", pipeline="raster_reproject.gdalg.json", output="out.tif", arguments=
↪{"edit[0].metadata": "before=modified"})
```

Placeholder dataset name `_PIPE_`

Added in version 3.13.

By default, in a pipeline step that accepts multiple input dataset arguments, the first positional argument, `input`, is implicitly set to the output dataset from the previous step. In some cases, it might be desirable to pipe the output dataset from the previous step into one of the other input dataset arguments instead.

This can be achieved by using the placeholder dataset name `_PIPE_` (PIPE with leading and trailing underscore character) as the value for the alternate dataset argument, while explicitly specifying the input positional dataset argument.

Example 2: Summarize mean elevation within 200m of points of interest

```
gdal pipeline read points.geojson ! buffer 200 ! \
    zonal-stats \
      --input dem.tif
      --zones _PIPE_ \
      --stat mean ! \
    write \
      --output-format CSV \
      --output /vsistdout/
```

It is also possible to achieve the same result by using a input nested pipeline as described below.

Nested pipeline

It is possible to create "nested pipelines", i.e. pipelines inside pipelines.

A nested pipeline is delimited by square brackets (`[` and `]`) surrounded by a space character.

There are 2 kinds of nested pipelines:

- input nested pipelines: where the result dataset of the nested pipeline is used as the input dataset for an argument of the main pipeline.
- output nested pipelines: where the output of a step of the main pipeline is used as the input of the nested pipeline in a following step. Output nested pipelines can only be used with the `tee` step.

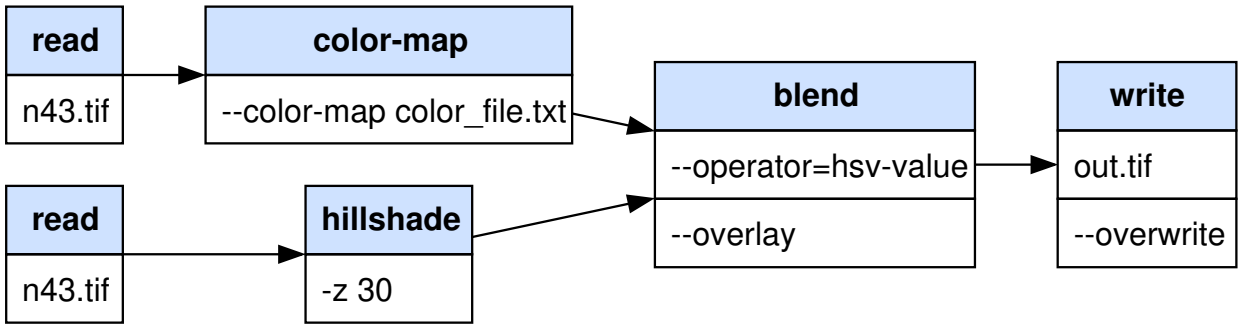
Input nested pipeline

Wherever an input dataset is expected as an auxiliary dataset, it is possible to specify it as the result of a nested pipeline. The content of an input nested pipeline is identical to the outer pipeline, except it must not end with an output-generating step like `info`, `tile` or `write`

Example 3: Combine the output of shaded relief map and hypsometric rendering on a DEM to create a colorized shaded relief map.

```
$ gdal pipeline read n43.tif ! \
                color-map --color-map color_file.txt ! \
                blend --operator=hsv-value --overlay \
                    [ read n43.tif ! hillshade -z 30 ] ! \
                write out.tif --overwrite
```

In the above example, the value of the `overlay` argument of the `blend` step is set as the output of the nested pipeline `read n43.tif ! hillshade -z 30`.



Output nested pipeline

The `tee` step in a pipeline forwards the input dataset as its output, and additionally executes one or several nested pipelines that take this input dataset as input and do other processing to eventually write the output of that processing. The first step of a `tee` output nested pipeline *must* not be `read`, `calc`, `concat`, `mosaic` or `stack`, and its last step *must* be `write` or `tile`. The `tee` operator can be either used in the middle of a pipeline or as its last step.

The below example shows an example where the `tee` operator executes two output nested pipelines.

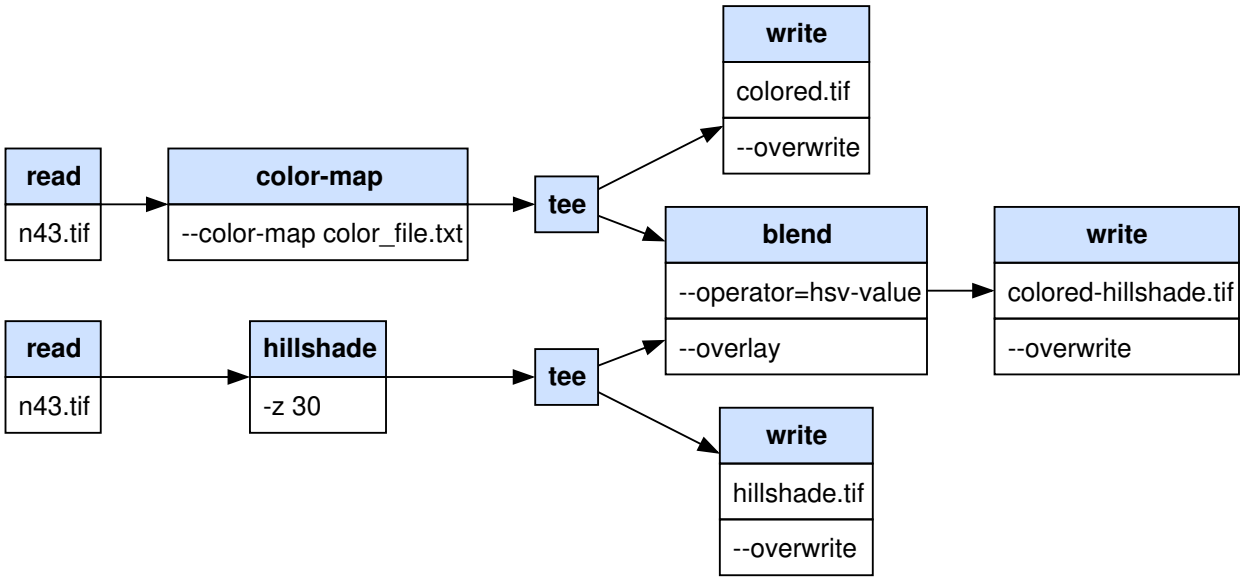
Example 4: Split the content of a "cities" layer according to whether its population is below or above 1 million.

```
$ gdal pipeline read cities.gpkg ! \
        tee [ filter --where "pop < 1e6" ! write small_cities.gpkg ] \
            [ filter --where "pop >= 1e6" ! write big_cities.gpkg ]
```

The below example shows a more complicated use case, including two occurrences of `tee`, with one of them being an output nested pipeline inside an input nested pipeline.

Example 5: Combine the output of shaded relief map and hypsometric rendering on a DEM to create a colorized shaded relief map, and write intermediate hillshade and colorized dataset

```
$ gdal pipeline read n43.tif ! \
                color-map --color-map color_file.txt ! \
                tee [ write colored.tif --overwrite ] ! \
                blend --operator=hsv-value --overlay \
                    [ read n43.tif ! hillshade -z 30  ! tee [ write hillshade.tif --
↪overwrite ] ] ! \
                write colored-hillshade.tif --overwrite
```



Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 6: Compute the footprint of a raster and apply a buffer on the footprint

```
$ gdal pipeline ! read in.tif ! footprint ! buffer 20 ! write out.gpkg --overwrite
```

Example 7: Rasterize and reproject

```
$ gdal pipeline ! read in.gpkg ! rasterize --size 1000,1000 ! reproject --output-crs␣
↪EPSG:4326 ! write out.tif --overwrite
```

Example 8: Use an existing pipeline that rasterizes and reprojects, but change its input file and target CRS, and specify the output file

```
$ gdal pipeline raster_reproject.gdalg.json --input=my.gpkg --output=out.tif --output-
↪crs=EPSG:32631
```

Example 9: Buffer a line dataset to create a new polygon dataset

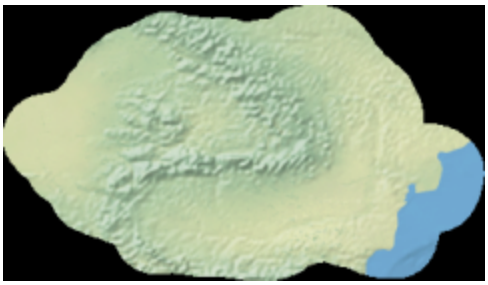
This example uses a `lines.gpkg` dataset containing a single layer named `lines`, with a geometry field named `geom` and an integer attribute named `width`. The value of this attribute is used as the buffer distance for each feature.

```
gdal vector pipeline \
    ! read lines.gpkg \
    ! sql "SELECT fid, ST_Buffer(geom, width) AS geom FROM lines" \
    ! set-geom-type --geometry-type Polygon \
    ! write buffered-lines.gpkg --output-layer=BufferedLines --overwrite --overwrite-
↪layer
```

> **Note**
>
> When creating derived geometries using SQL, avoid using `SELECT *`. Including the original geometry field will result in multiple geometry columns in the output. Instead, explicitly list the required attributes and return a single geometry column.

Example 10: Clip a raster dataset using a nested vector pipeline

This pipeline uses a raster input dataset, and clips it using a buffered geometry coming from a nested vector pipeline. The resulting clipped raster is then resized and written to disk, and shown in the image below.

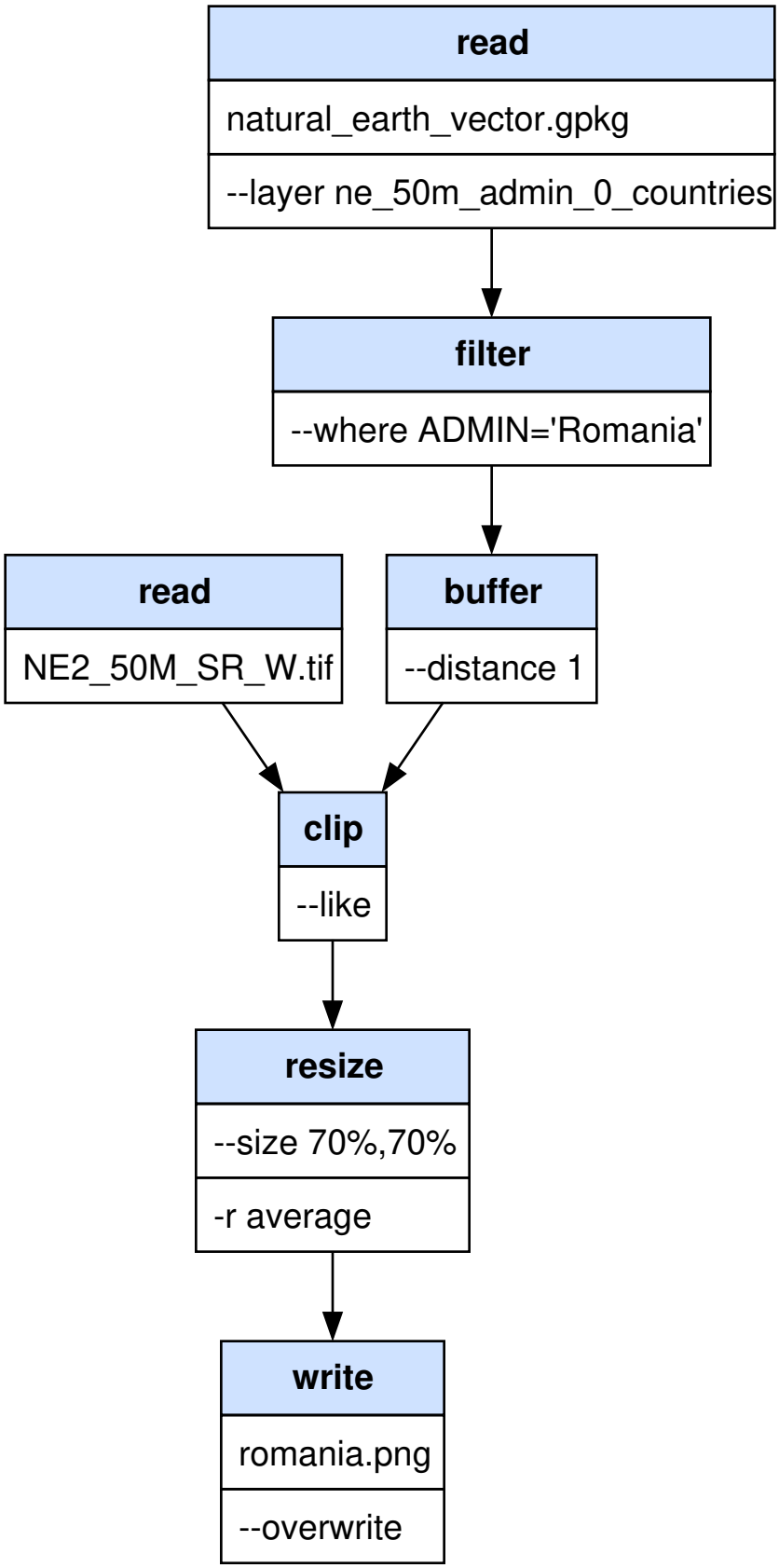


Bash

PowerShell

```
gdal pipeline \
    ! read "NE2_50M_SR_W.tif" \
    ! clip --like [ read natural_earth_vector.gpkg --layer "ne_50m_admin_0_countries" !␣
↪filter --where "ADMIN='Romania'" ! buffer --distance=1 ] \
    ! resize --size=70%,70% -r average \
    ! write romania.png --overwrite
```

```
gdal pipeline `
    ! read "NE2_50M_SR_W.tif" `
    ! clip --like [ read natural_earth_vector.gpkg --layer "ne_50m_admin_0_countries" !␣
↪filter --where "ADMIN='Romania'" ! buffer --distance=1 ] `
    ! resize --size=70%,70% -r average `
    ! write romania.png --overwrite
```


read
natural_earth_vector.gpkg
--layer ne_50m_admin_0_countries
filter
--where ADMIN='Romania'
read
NE2_50M_SR_W.tif
buffer
--distance 1
clip
--like
resize
--size 70%,70%
-r average
write
romania.png
--overwrite

Example 11: Clip a raster dataset with vector features using _PIPE_

This pipeline produces the same result as *Example 10*, but instead of using a nested pipeline for the clipping geometry, it uses the special placeholder dataset name *__PIPE__* to pass the output of the buffering step to `clip`.

Bash

PowerShell

```
gdal pipeline \
  ! read natural_earth_vector.gpkg --layer "ne_50m_admin_0_countries" \
  ! filter --where "ADMIN='Romania'" \
  ! buffer --distance=1 \
  ! clip --input "NE2_50M_SR_W.tif" --like _PIPE_ \
  ! resize --size=70%,70% -r average \
  ! write romania.png --overwrite
```

```
gdal pipeline `
  ! read natural_earth_vector.gpkg --layer "ne_50m_admin_0_countries" `
  ! filter --where "ADMIN='Romania'" `
  ! buffer --distance=1 `
  ! clip --input "NE2_50M_SR_W.tif" --like _PIPE_ `
  ! resize --size=70%,70% -r average `
  ! write romania.png --overwrite
```

5.1.2.4 `gdal external`

Added in version 3.13.

Description

`external` can be used as a step (in any position) in a generic, raster or vector pipeline to execute an arbitrary external process.

For security reasons (in particular when embedded in GDALG pipelines), its execution requires the `GDAL_ENABLE_EXTERNAL` configuration option to be set to YES.

The command line executed by this step can accept the following optional placeholders:

- `<INPUT>` that is replaced by the name of a temporary file, generated from the input of the previous step. The format used by default to materialize that file is GeoTIFF for raster and GeoPackage for vector, unless the *`--input-format`* is specified.
- `<OUTPUT>` that is replaced by the name of a temporary file, generated by the external command. The format used by default to materialize that file is GeoTIFF for raster and GeoPackage for vector, unless the *`--output-format`* is specified. Upon the completion of the external command, the temporary file is opened, and passed to the next step of the pipeline.
- `<INPUT-OUTPUT>` that is used for an external program that updates in place a dataset. The format used by default to materialize that file is GeoTIFF for raster and GeoPackage for vector, unless the *`--input-format`* is specified.

The directory where temporary files are created can be set with the *`CPL_TMPDIR`* configuration option.

If no `<OUTPUT>` or `<INPUT-OUTPUT>` placeholder is specified, the output of the step before `external` is reused as the output of `external`.

Synopsis

```
* external [OPTIONS] <COMMAND>
-----------------------------

Execute an external program as a step of a pipeline

Positional arguments:
  --command <COMMAND>                                External command, optionally with
↪<INPUT> and/or <OUTPUT> or <INPUT-OUTPUT> placeholders [required]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                  Input formats
```

Program-Specific Options

--command <COMMAND>

External command, optionally with <INPUT> and/or <OUTPUT> or <INPUT-OUTPUT> placeholders [required]

Standard Options

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Pipeline running a QGIS processing algorithm to smooth a DEM before computing the hillshade.

```
$ gdal pipeline --config GDAL_ENABLE_EXTERNAL=YES \
    read dem.tif ! \
    external "qgis_process run native:rasterfeaturepreservingsmoothing -- INPUT=<INPUT>␣
↪OUTPUT=<OUTPUT>" ! \
    hillshade ! \
    write out.tif
```

5.1.3 Raster commands

5.1.3.1 gdal raster

Added in version 3.11.

Synopsis

```
Usage: gdal raster <SUBCOMMAND> [OPTIONS]
where <SUBCOMMAND> is one of:
  - as-features:     Create features from pixels of a raster dataset
  - aspect:          Generate an aspect map
  - blend:           Blend/compose two raster datasets
  - calc:            Perform raster algebra
  - clean-collar:    Clean the collar of a raster dataset, removing noise.
  - clip:            Clip a raster dataset.
  - color-map:       Generate a RGB or RGBA dataset from a single band, using a color map
  - compare:         Compare two raster datasets.
  - contour:         Creates a vector contour from a raster elevation model (DEM).
  - convert:         Convert a raster dataset.
  - create:          Create a new raster dataset.
  - edit:            Edit a raster dataset.
  - fill-nodata:     Fill nodata raster regions by interpolation from edges.
  - footprint:       Compute the footprint of a raster dataset.
  - hillshade:       Generate a shaded relief map
  - index:           Create a vector index of raster datasets.
  - info:            Return information on a raster dataset.
  - mosaic:          Build a mosaic, either virtual (VRT) or materialized.
  - neighbors:       Compute the value of each pixel from its neighbors (focal␣
↪statistics) (alias: neighbours)
  - nodata-to-alpha: Replace nodata value(s) with an alpha band.
  - overview:        Manage overviews of a raster dataset.
  - pansharpen:      Perform a pansharpen operation.
  - pipeline:        Process a raster dataset applying several steps.
  - pixel-info:      Return information on a pixel of a raster dataset.
  - polygonize:      Create a polygon feature dataset from a raster band.
  - proximity:       Produces a raster proximity map.
  - reclassify:      Reclassify values in a raster dataset
  - reproject:       Reproject a raster dataset.
  - resize:          Resize a raster dataset without changing the georeferenced extents.
  - rgb-to-palette:  Convert a RGB image into a pseudo-color / paletted image.
  - roughness:       Generate a roughness map
  - scale:           Scale the values of the bands of a raster dataset.
  - select:          Select a subset of bands from a raster dataset.
  - set-type:        Modify the data type of bands of a raster dataset.
  - sieve:           Remove small polygons from a raster dataset.
  - slope:           Generate a slope map
  - stack:           Combine together input bands into a multi-band output, either␣
↪virtual (VRT) or materialized.
  - tile:            Generate tiles in separate files from a raster dataset.
  - tpi:             Generate a Topographic Position Index (TPI) map
  - tri:             Generate a Terrain Ruggedness Index (TRI) map
  - unscale:         Convert scaled values of a raster dataset into unscaled values.
  - update:          Update the destination raster with the content of the input one.
```

(continues on next page)

(continued from previous page)

```
  - viewshed:        Compute the viewshed of a raster dataset.
  - zonal-stats:     Calculate raster zonal statistics

Common Options:
  -h, --help              Display help message and exit
  --json-usage            Display usage as JSON document and exit
  --config <KEY>=<VALUE>  Configuration option [may be repeated]

Options:
  --drivers               Display raster driver list as JSON document
```

Available sub-commands

- *gdal raster info*
- *gdal raster aspect*
- *gdal raster blend*
- *gdal raster calc*
- *gdal raster clean-collar*
- *gdal raster clip*
- *gdal raster color-map*
- *gdal raster compare*
- *gdal raster convert*
- *gdal raster create*
- *gdal raster footprint*
- *gdal raster fill-nodata*
- *gdal raster hillshade*
- *gdal raster index*
- *gdal raster mosaic*
- *gdal raster neighbors*
- *gdal raster nodata-to-alpha*
- *gdal raster overview*
- *gdal raster pipeline*
- *gdal raster pixel-info*
- *gdal raster polygonize*
- *gdal raster proximity*
- *gdal raster reproject*
- *gdal raster resize*
- *gdal raster rgb-to-palette*
- *gdal raster roughness*

- *gdal raster scale*
- *gdal raster select*
- *gdal raster sieve*
- *gdal raster slope*
- *gdal raster stack*
- *gdal raster tile*
- *gdal raster tpi*
- *gdal raster tri*
- *gdal raster unscale*
- *gdal raster update*
- *gdal raster viewshed*

Examples

Example 1: Getting information on the file utm.tif (with text output)

```
$ gdal raster info utm.tif
```

Example 2: Converting file utm.tif to GeoPackage raster

```
$ gdal raster convert utm.tif utm.gpkg
```

Example 3: Getting the list of raster drivers (with JSON output)

```
$ gdal raster --drivers
```

5.1.3.2 gdal raster info

Added in version 3.11.

Synopsis

```
Usage: gdal raster info [OPTIONS] <INPUT>

Return information on a raster dataset.

Positional arguments:
  -i, --dataset, --input <INPUT>                      Input raster dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                          Display help message and exit
  --json-usage                                        Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                              Configuration option [may be␣
↪repeated]
```

(continues on next page)

(continued from previous page)

```
Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format. OUTPUT-
↪FORMAT=json|text
  --mm, --min-max                                      Compute minimum and maximum value
  --stats                                              Retrieve or compute statistics,␣
↪using all pixels
                                                       Mutually exclusive with --approx-
↪stats
  --approx-stats                                       Retrieve or compute statistics,␣
↪using a subset of pixels
                                                       Mutually exclusive with --stats
  --hist                                               Retrieve or compute histogram

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                  Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                    Open options [may be repeated]␣
↪[not available in pipelines]
  --no-gcp                                             Suppress ground control points␣
↪list printing
  --no-md                                              Suppress metadata printing
  --no-ct                                              Suppress color table printing
  --no-fl                                              Suppress file list printing
  --checksum                                           Compute pixel checksum
  --list-metadata-domains, --list-mdd                  List all metadata domains␣
↪available for the dataset
  --mdd, --metadata-domain <METADATA-DOMAIN>           Report metadata for the specified␣
↪domain. 'all' can be used to report metadata in all domains

Esoteric Options:
  --no-nodata                                          Suppress retrieving nodata value
  --no-mask                                            Suppress mask band information
  --subdataset <SUBDATASET>                            Use subdataset of specified index␣
↪(starting at 1), instead of the source dataset itself
  --crs-format <CRS-FORMAT>                            Which format to use to report CRS.
↪ CRS-FORMAT=AUTO|WKT2|PROJJSON (default: AUTO)
```

Description

gdal raster info lists various information about a GDAL supported raster dataset, and returns them on the standard output stream when used from the command line, or in the `output` parameter when used from the API.

The following items will be reported (when known):

- The format driver used to access the file.
- Raster size (in pixels and lines).
- The coordinate system for the file (in short form AUTH_NAME:CODE when possible, or otherwise OGC WKT2:2019).
- The geotransform associated with the file (rotational coefficients are currently not reported).
- Corner coordinates in georeferenced, and if possible lat/long based on the full geotransform (but not GCPs).

- Ground control points.
- File wide (including subdatasets) metadata.
- Band data types.
- Band color interpretations.
- Band block size.
- Band descriptions.
- Band min/max values (internally known and possibly computed).
- Band checksum (if computation asked).
- Band NODATA value.
- Band overview resolutions available.
- Band unit type (i.e.. "meters" or "feet" for elevation bands).
- Band pseudo-color tables.

Starting with GDAL 3.12, **gdal raster info** can be used as the last step of a pipeline.

The following options are available:

Program-Specific Options

--approx-stats

Read and display image statistics. Force computation if no statistics are stored in an image. However, they may be computed based on overviews or a subset of all tiles. Useful if you are in a hurry and don't need precise stats.

--checksum

Force computation of the checksum for each band in the dataset.

-f, --of, --format, --output-format `json|text`

Which output format to use. Default is JSON, and starting with GDAL 3.12, text when invoked from command line.

--crs-format `AUTO|WKT2|PROJJSON`

Added in version 3.13.

Which format to use to report the CRS. In AUTO default mode, if the CRS can be captured with an authority name and code (known of PROJ), only a summary of the CRS, including its name, ID, type and area of use will be reported. Otherwise a full WKT2:2019 definition will be reported.

> **Note**
>
> *--crs-format* can only be set when *--output-format* is set to `text`. The JSON text format includes both WKT2 and PROJJSON.

--hist

Report histogram information for all bands.

--list-mdd

List all metadata domains available for the dataset.

--mdd, --metadata-domain <domain>|all

adds metadata using:

domain Report metadata for the specified domain.

all Report metadata for all domains.

--mm, --min-max

Force computation of the actual min/max values for each band in the dataset.

--no-ct

Suppress printing of color table.

--no-fl

Only display the first file of the file list.

--no-gcp

Suppress ground control points list printing. It may be useful for datasets with huge amount of GCPs, such as L1B AVHRR or HDF4 MODIS which contain thousands of them.

--no-mask

Suppress band mask printing. Is implied if *--no-nodata* is specified.

--no-md

Suppress metadata printing. Some datasets may contain a lot of metadata strings.

--no-nodata

Suppress nodata printing. Implies *--no-mask*.

Can be useful for example when querying a remove GRIB2 dataset that has an index .idx side-car file, together with *--no-md*

--stats

Read and display image statistics. Force computation if no statistics are stored in an image.

--subdataset <n>

If the input dataset contains several subdatasets read and display a subdataset with specified n number (starting from 1). This is an alternative of giving the full subdataset name.

Standard Options

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Getting information on the file utmsmall.tif as text output

```
$ gdal raster info utmsmall.tif
Driver: GTiff/GeoTIFF
Files: utmsmall.tif
Size is 100, 100
Coordinate Reference System:
  - name: NAD27 / UTM zone 11N
  - ID: EPSG:26711
  - type: Projected
  - projection type: Transverse Mercator
  - units: metre
Data axis to CRS axis mapping: 1,2
Origin = (440720.000000000000000,3751320.000000000000000)
Pixel Size = (60.000000000000000,-60.000000000000000)
Metadata:
  AREA_OR_POINT=Area
Image Structure Metadata:
  INTERLEAVE=BAND
Corner Coordinates:
Upper Left  (  440720.000, 3751320.000) (117d38'28.21"W, 33d54' 8.47"N)
Lower Left  (  440720.000, 3745320.000) (117d38'26.75"W, 33d50'53.66"N)
Upper Right (  446720.000, 3751320.000) (117d34'34.59"W, 33d54' 9.62"N)
Lower Right (  446720.000, 3745320.000) (117d34'33.28"W, 33d50'54.82"N)
Center      (  443720.000, 3748320.000) (117d36'30.71"W, 33d52'31.66"N)
Band 1 Block=100x81 Type=Byte, ColorInterp=Gray
```

Example 2: Getting information on the file utmsmall.tif as JSON output, including statistics

```
$ gdal raster info --format=JSON --stats utmsmall.tif
{
  "description":"utmsmall.tif",
  "driverShortName":"GTiff",
  "driverLongName":"GeoTIFF",
  "files":[
    "utmsmall.tif"
  ],
  "size":[
    100,
    100
  ],
  "coordinateSystem":{
    "wkt":"PROJCRS[\"NAD27 / UTM zone 11N\",\n    BASEGEOGCRS[\"NAD27\",\n        DATUM[\
↪"North American Datum 1927\",\n            ELLIPSOID[\"Clarke 1866\",6378206.4,294.
↪978698213898,\n                LENGTHUNIT[\"metre\",1]]],\n        PRIMEM[\"Greenwich\
↪",0,\n            ANGLEUNIT[\"degree\",0.0174532925199433]],\n        ID[\"EPSG\",
↪4267]],\n    CONVERSION[\"Transverse Mercator\",\n        METHOD[\"Transverse Mercator\
↪",\n            ID[\"EPSG\",9807]],\n        PARAMETER[\"Latitude of natural origin\",
↪0,\n            ANGLEUNIT[\"degree\",0.0174532925199433],\n            ID[\"EPSG\",
↪8801]],\n        PARAMETER[\"Longitude of natural origin\",-117,\n
```

(continues on next page)

(continued from previous page)

```
↪ANGLEUNIT[\"degree\",0.0174532925199433],\n            ID[\"EPSG\",8802]],\n        ␣
↪PARAMETER[\"Scale factor at natural origin\",0.9996,\n            SCALEUNIT[\"unity\",
↪1],\n            ID[\"EPSG\",8805]],\n        PARAMETER[\"False easting\",500000,\n    ␣
↪        LENGTHUNIT[\"metre\",1],\n            ID[\"EPSG\",8806]],\n        PARAMETER[\
↪"False northing\",0,\n            LENGTHUNIT[\"metre\",1],\n            ID[\"EPSG\",
↪8807]]],\n    CS[Cartesian,2],\n        AXIS[\"easting\",east,\n            ORDER[1],\
↪n            LENGTHUNIT[\"metre\",1]],\n        AXIS[\"northing\",north,\n            ␣
↪ORDER[2],\n            LENGTHUNIT[\"metre\",1]],\n    ID[\"EPSG\",26711]]",
    "dataAxisToSRSAxisMapping":[
      1,
      2
    ]
  },
  "geoTransform":[
    440720.0,
    60.0,
    0.0,
    3751320.0,
    0.0,
    -60.0
  ],
  "metadata":{
    "":{
      "AREA_OR_POINT":"Area"
    },
    "IMAGE_STRUCTURE":{
      "INTERLEAVE":"BAND"
    }
  },
  "cornerCoordinates":{
    "upperLeft":[
      440720.0,
      3751320.0
    ],
    "lowerLeft":[
      440720.0,
      3745320.0
    ],
    "lowerRight":[
      446720.0,
      3745320.0
    ],
    "upperRight":[
      446720.0,
      3751320.0
    ],
    "center":[
      443720.0,
      3748320.0
    ]
  },
  "wgs84Extent":{
```

(continues on next page)

(continued from previous page)


```
    "type":"Polygon",
    "coordinates":[
      [
        [
          -117.6420428,
          33.9023684
        ],
        [
          -117.6416377,
          33.848257
        ],
        [
          -117.5767844,
          33.8485775
        ],
        [
          -117.5771485,
          33.9026895
        ],
        [
          -117.6420428,
          33.9023684
        ]
      ]
    ]
  },
  "bands":[
    {
      "band":1,
      "block":[
        100,
        81
      ],
      "type":"Byte",
      "colorInterpretation":"Gray",
      "minimum":0.0,
      "maximum":255.0,
      "mean":154.621,
      "stdDev":54.251,
      "metadata":{
        "":{
          "STATISTICS_MINIMUM":"0",
          "STATISTICS_MAXIMUM":"255",
          "STATISTICS_MEAN":"154.6212",
          "STATISTICS_STDDEV":"54.250980733624",
          "STATISTICS_VALID_PERCENT":"100"
        }
      }
    }
  ],
  "stac":{
    "proj:shape":[
```


(continues on next page)

(continued from previous page)

```
      100,
      100
    ],
    "proj:wkt2":"PROJCRS[\"NAD27 / UTM zone 11N\",\n    BASEGEOGCRS[\"NAD27\",\n
↪DATUM[\"North American Datum 1927\",\n            ELLIPSOID[\"Clarke 1866\",6378206.4,
↪294.978698213898,\n                LENGTHUNIT[\"metre\",1]]],\n        PRIMEM[\
↪"Greenwich\",0,\n            ANGLEUNIT[\"degree\",0.0174532925199433]],\n        ID[\
↪"EPSG\",4267]],\n    CONVERSION[\"Transverse Mercator\",\n        METHOD[\"Transverse
↪Mercator\",\n            ID[\"EPSG\",9807]],\n        PARAMETER[\"Latitude of natural
↪origin\",0,\n            ANGLEUNIT[\"degree\",0.0174532925199433],\n            ID[\
↪"EPSG\",8801]],\n        PARAMETER[\"Longitude of natural origin\",-117,\n
↪ANGLEUNIT[\"degree\",0.0174532925199433],\n            ID[\"EPSG\",8802]],\n
↪PARAMETER[\"Scale factor at natural origin\",0.9996,\n            SCALEUNIT[\"unity\",
↪1],\n            ID[\"EPSG\",8805]],\n        PARAMETER[\"False easting\",500000,\n
↪        LENGTHUNIT[\"metre\",1],\n            ID[\"EPSG\",8806]],\n        PARAMETER[\
↪"False northing\",0,\n            LENGTHUNIT[\"metre\",1],\n            ID[\"EPSG\",
↪8807]]],\n    CS[Cartesian,2],\n        AXIS[\"easting\",east,\n            ORDER[1],\
↪n            LENGTHUNIT[\"metre\",1]],\n        AXIS[\"northing\",north,\n
↪ORDER[2],\n            LENGTHUNIT[\"metre\",1]],\n    ID[\"EPSG\",26711]]",
    "proj:epsg":26711,
    "proj:projjson":{
      "$schema":"https://proj.org/schemas/v0.7/projjson.schema.json",
      "type":"ProjectedCRS",
      "name":"NAD27 / UTM zone 11N",
      "base_crs":{
        "name":"NAD27",
        "datum":{
          "type":"GeodeticReferenceFrame",
          "name":"North American Datum 1927",
          "ellipsoid":{
            "name":"Clarke 1866",
            "semi_major_axis":6378206.4,
            "inverse_flattening":294.978698213898
          }
        },
        "coordinate_system":{
          "subtype":"ellipsoidal",
          "axis":[
            {
              "name":"Geodetic latitude",
              "abbreviation":"Lat",
              "direction":"north",
              "unit":"degree"
            },
            {
              "name":"Geodetic longitude",
              "abbreviation":"Lon",
              "direction":"east",
              "unit":"degree"
            }
          ]
        },
```

(continues on next page)

(continued from previous page)

```
        "id":{
          "authority":"EPSG",
          "code":4267
        }
      },
      "conversion":{
        "name":"Transverse Mercator",
        "method":{
          "name":"Transverse Mercator",
          "id":{
            "authority":"EPSG",
            "code":9807
          }
        },
        "parameters":[
          {
            "name":"Latitude of natural origin",
            "value":0,
            "unit":"degree",
            "id":{
              "authority":"EPSG",
              "code":8801
            }
          },
          {
            "name":"Longitude of natural origin",
            "value":-117,
            "unit":"degree",
            "id":{
              "authority":"EPSG",
              "code":8802
            }
          },
          {
            "name":"Scale factor at natural origin",
            "value":0.9996,
            "unit":"unity",
            "id":{
              "authority":"EPSG",
              "code":8805
            }
          },
          {
            "name":"False easting",
            "value":500000,
            "unit":"metre",
            "id":{
              "authority":"EPSG",
              "code":8806
            }
          },
          {
```

(continues on next page)

(continued from previous page)


```
            "name":"False northing",
            "value":0,
            "unit":"metre",
            "id":{
              "authority":"EPSG",
              "code":8807
            }
          }
        ]
      },
      "coordinate_system":{
        "subtype":"Cartesian",
        "axis":[
          {
            "name":"Easting",
            "abbreviation":"",
            "direction":"east",
            "unit":"metre"
          },
          {
            "name":"Northing",
            "abbreviation":"",
            "direction":"north",
            "unit":"metre"
          }
        ]
      },
      "id":{
        "authority":"EPSG",
        "code":26711
      }
    },
    "proj:transform":[
      60.0,
      0.0,
      440720.0,
      0.0,
      -60.0,
      3751320.0
    ],
    "raster:bands":[
      {
        "data_type":"uint8",
        "stats":{
          "minimum":0.0,
          "maximum":255.0,
          "mean":154.621,
          "stddev":54.251
        }
      }
    ],
    "eo:bands":[
```


(continues on next page)

(continued from previous page)

```
      {
        "name":"b1",
        "description":"Gray"
      }
    ]
  }
}
```

Example 3: Check if a remote GeoTIFF is a COG

Search for `LAYOUT=COG` in the `Image Structure Metadata`. The example below uses jq and returns *true* if the remote image is a COG.

Bash

PowerShell

```
$ gdal raster info https://sentinel-cogs.s3.us-west-2.amazonaws.com/sentinel-s2-l2a-cogs/
↪36/Q/WD/2020/7/S2A_36QWD_20200701_0_L2A/TCI.tif --of=JSON \
| jq '.metadata.IMAGE_STRUCTURE.LAYOUT == "COG"'
```

```
> gdal raster info https://sentinel-cogs.s3.us-west-2.amazonaws.com/sentinel-s2-l2a-cogs/
↪36/Q/WD/2020/7/S2A_36QWD_20200701_0_L2A/TCI.tif --of=JSON `
| jq '.metadata.IMAGE_STRUCTURE.LAYOUT == \"COG\"'
```

Example 4: Return the CRS used by the raster using jq

Bash

PowerShell

```
gdal raster info in.tif --of json | jq -r '.stac."proj:projjson".id | .authority + ":" +␣
↪(.code|tostring)'
```

```
gdal raster info in.tif --of json | jq -r '.stac.\"proj:projjson\".id | .authority + \":\
↪" + (.code|tostring)'
```

5.1.3.3 gdal raster as-features

Added in version 3.12.

Synopsis

```
Usage: gdal raster as-features [OPTIONS] <INPUT> <OUTPUT>

Create features from pixels of a raster dataset

Positional arguments:
  -i, --input <INPUT>                                    Input raster datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]
```

(continues on next page)

(continued from previous page)

```
Common Options:
  -h, --help                                          Display help message and exit
  --json-usage                                        Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                              Configuration option [may be␣
↪repeated]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>               Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>        Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                         Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                            Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                   Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                            Whether appending to existing␣
↪layer is allowed [not available in pipelines]
  --output-layer <OUTPUT-LAYER>                       Output layer name [not available␣
↪in pipelines]
  -b, --band <BAND>                                   Input band(s) (1-based index)␣
↪[may be repeated]
  --geometry-type <GEOMETRY-TYPE>                     Geometry type. GEOMETRY-
↪TYPE=none|point|polygon (default: none)
  --skip-nodata                                       Omit NoData pixels from the result
  --include-xy                                        Include fields for cell center␣
↪coordinates
  --include-row-col                                   Include columns for row and column

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                 Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                   Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>     Output open options [may be␣
↪repeated] [not available in pipelines]
```

Description

gdal raster as-features creates features representing the pixels in a raster dataset. Features may be created as polygons, points, or with no geometry at all. Unlike *gdal raster polygonize*, adjacent pixels having the same values are not combined.

This algorithm can be part of a *gdal pipeline*.

Program-Specific Options

-b, --band <band>

Specifies the bands for which pixel values should be added as fields of the created features. By default, values will be added for all bands.

--geometry-type

Specifies the geometry type of the created features. Options available are "none" (default), "point", or "polygon".

--include-row-col

If set, ROW and COL fields will be added with the cell positions.

--include-xy

If set, CENTER_X and CENTER_Y fields will be added with the center coordinates of each pixel.

--output-layer

Provides a name for the output vector layer. Defaults to "pixels".

--skip-nodata

If set, no features will be emitted for pixels equal to the NoData value.

Standard Options

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--update

Whether to open an existing output dataset in update mode.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Create points at the center of pixels having a value less than 150

```
gdal pipeline read input.tif !
         reclassify -m "[-inf, 150)=1; DEFAULT=NO_DATA" !
         as-features --geometry-type point --skip-nodata !
         write out.shp
```

Example 2: Create a polygon grid dividing the globe into 1-degree chunks

```
gdal pipeline create --bbox -180,-90,180,90 --size 360,180 !
         as-features --geometry-type polygon !
         write grid.shp
```

5.1.3.4 gdal raster aspect

Added in version 3.11.

Synopsis

```
Usage: gdal raster aspect [OPTIONS] <INPUT> <OUTPUT>

Generate an aspect map

Positional arguments:
  -i, --input <INPUT>                                    Input raster datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output raster dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                         Mutually exclusive with --append
  --append                                               Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                         Mutually exclusive with --
↪overwrite
  -b, --band <BAND>                                      Input band (1-based index)␣
↪(default: 1)
  --convention <CONVENTION>                              Convention for output angles.␣
↪CONVENTION=azimuth|trigonometric-angle (default: azimuth)
  --gradient-alg <GRADIENT-ALG>                          Algorithm used to compute terrain␣
↪gradient. GRADIENT-ALG=Horn|ZevenbergenThorne (default: Horn)
  --zero-for-flat                                        Whether to output zero for flat␣
↪areas
  --no-edges                                             Do not try to interpolate values␣
↪at dataset edges or close to nodata values

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                    Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                      Open options [may be repeated]␣
↪[not available in pipelines]
```

Description

gdal raster aspect generates an aspect map, from any GDAL-supported elevation raster.

This subcommand is also available as a potential step of *gdal raster pipeline*

It outputs a 32-bit float raster with values between 0° and 360° representing the azimuth that slopes are facing. The definition of the azimuth is such that: - 0° means that the slope is facing the North, - 90° it's facing the East, - 180° it's facing the South - and 270° it's facing the West (provided that the top of your input raster is north oriented).

The aspect value -9999 is used as the nodata value to indicate undefined aspect in flat areas with slope=0.

A nodata value in the target dataset will also be emitted if at least one pixel set to the nodata value is found in the 3x3 window centered around each source pixel. By default, the algorithm will compute values at image edges or if a nodata value is found in the 3x3 window, by interpolating missing values, unless `--no-edges` is specified, in which case a 1-pixel border around the image will be set with the nodata value.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

-b, --band <BAND>

Index (starting at 1) of the band to which the aspect must be computed.

--convention `azimuth|trigonometric-angle`

Convention for output angles.

Defaults to `azimuth`, that is to say: - 0° means that the slope is facing the North, - 90° it's facing the East, - 180° it's facing the South - and 270° it's facing the West (provided that the top of your input raster is north oriented).

If set to `trigonometric-angle`, - 0° means that the slope is facing the East, - 90° it's facing the North, - 180° it's facing the West - and 270° it's facing the South

--gradient-alg `Horn|ZevenbergenThorne`

Algorithm used to compute terrain gradient. The default is `Horn`. The literature suggests Zevenbergen & Thorne to be more suited to smooth landscapes, whereas Horn's formula to perform better on rougher terrain.

--no-edges

Do not try to interpolate values at dataset edges or close to nodata values

--zero-for-flat

Whether to output zero for flat areas. By default, flat areas where the slope is null will be assigned a nodata value (-9999). When setting this option, they are set to 0.

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Generates an aspect map from a DTED0 file.

```
$ gdal raster aspect n43.dt0 out.tif --overwrite
```

5.1.3.5 gdal raster blend

Added in version 3.12.

Synopsis

```
Usage: gdal raster blend [OPTIONS] <COLOR-INPUT> <OVERLAY> <OUTPUT>

Blend/compose two raster datasets

Positional arguments:
```

(continues on next page)

(continued from previous page)

```
  -i, --color-input, --input <COLOR-INPUT>             Input raster dataset [required]␣
↪[not available in pipelines]
  --overlay <OVERLAY>                                  Overlay dataset [required]
  -o, --output <OUTPUT>                                Output raster dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                           Display help message and exit
  --json-usage                                         Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                               Configuration option [may be␣
↪repeated]
  -q, --quiet                                          Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                Creation option [may be repeated]␣
↪[not available in pipelines]
  --overwrite                                          Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                       Mutually exclusive with --append
  --append                                             Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                       Mutually exclusive with --
↪overwrite
  --operator <OPERATOR>                                Composition operator.␣
↪OPERATOR=src-over|hsv-value|multiply|screen|overlay|hard-light|darken|lighten|color-
↪dodge|color-burn (default: src-over)
  --opacity <OPACITY>                                  Opacity percentage to apply to␣
↪the overlay dataset (0=fully transparent, 100=full use of overlay opacity) (default:␣
↪100)

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                  Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                    Open options [may be repeated]␣
↪[not available in pipelines]
```

Description

gdal raster blend allows the user to compose two raster datasets (of the same size) using a selected blending operation and an opacity setting.

This can be used for example to colorize a hillshade raster generated by *gdal raster hillshade* with an hypsometric rendering of a DEM generated by *gdal raster color-map* when using the `hsv-value` blending operator.

The more standard `src-over` blending operator, which does "alpha blending" is also available.

This subcommand is also available as a potential step of *gdal raster pipeline*

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

--input <INPUT>

Name of the dataset into which to blend the overlay dataset. Required.

When using the `hsv-value` blending operator, this must be a three-band or four-band Byte raster.

--opacity <OPACITY>

Opacity to use when blending the overlay dataset, as a percentage between 0 and 100. 0 means that the overlay dataset is considered fully transparent. 100 means that the overlay dataset is blended at the maximum of its alpha channel. Defaults to 100.

--operator `src-over|hsv-value|multiply|screen|overlay|hard-light|lighten|darken`

Select the blending operator, which defines how the overlay dataset is blended into the input dataset. Defaults to `src-over`.

See *Details of blending operations* below for details about each operator.

--overlay <OVERLAY>

Name of the overlay dataset. Required

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Details of blending operations

In the description of the blending operations, all values are normalized to [0,1] range and each channel value is premultiplied with the alpha channel value.

The following notations are used:

- $overlay_C$ is the value of one of the red, green or blue component of the overlay dataset,
- $overlay_A$ is the value of the alpha component of the overlay dataset premultiplied by `--opacity`,
- $input_C$ is the value of the corresponding component of the input dataset,
- $input_A$ is the value of the alpha component of the input dataset
- $output_C$ is the resulting component of the output dataset,
- $opacity$ is the value of `--opacity`

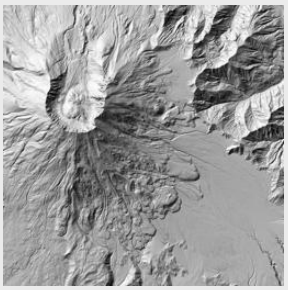
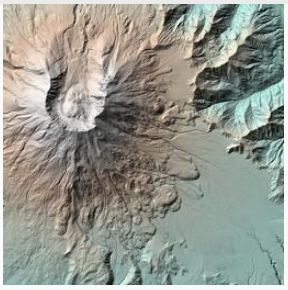
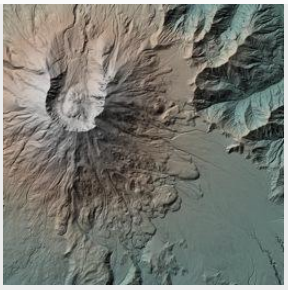
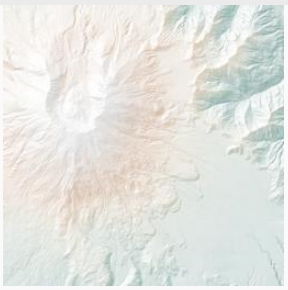
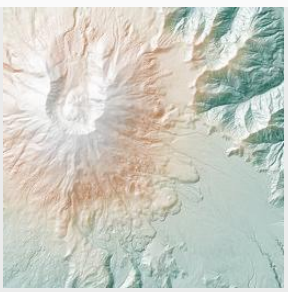
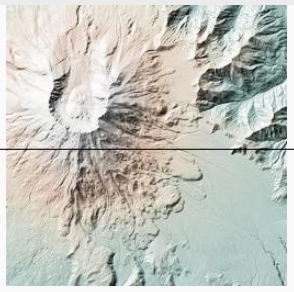
> **Note**
>
> For efficiency, the implementation uses integer arithmetic on [0,255] range rather than floating point arithmetic on [0,1] range. This can explain minor differences with the formulas described below.

For all operator (unless otherwise specified) the alpha channel is computed as:

$$output_A = overlay_A + input_A - overlay_A * input_A$$

The following blending operators are available:

Table 1: Overview of raster blend operators

Operator	Description	Example
`src-over`	Standard alpha blending (overlay composited over input)	
`hsv-value`	Uses value from overlay and hue/saturation from input	
`multiply`	Multiplies input and overlay colors	
`screen`	Complement then multiply colors	
`overlay`	Combines multiply and screen	
`hard-light`	Multiply or screen depending on overlay brightness	

- **`src-over` performs standard alpha blending, by compositing the overlay**
  dataset over the input dataset.

  The following formula is applied for each of the red, green and blue components:

  $$output_C * output_A = (overlay_C * overlay_A * opacity) + (input_C * input_A * (1 - overlay_A * opacity))$$

- **`hsv-value` creates a color with the value of the overlay and the hue**
  and saturation of the input. It performs the following steps:

  - read the RGB (Red,Green,Blue) components of the input dataset
  - transform the RGB components into the HSV (Hue,Saturation,Value) color space
  - compute the output value $output_V$, using the following formula:

    $$output_V = (overlay_V * overlay_A * opacity) + (input_V * (1 - overlay_A * opacity))$$

  If the overlay dataset is a RGB/RGBA dataset, $overlay_V$ is $max(overlay_R, overlay_G, overlay_B)$.

  - transform back (Hue,Saturation,:math:*output_{V}*) to RGB.

  If the the alpha channel of the input dataset is present, it is preserved unchanged.

- **`multiply` multiplies the input and overlay colors.**
  The resulting color is always at least as dark as either of the two constituent colors.

  The following formula is applied for each of the red, green and blue components:

  $$output_C = overlay_C * input_C + input_C * (1 - overlay_A) + overlay_C * (1 - input_A)$$

- `screen` The overlay and input are complemented and then multiplied. The resultant color is always at least as light as either of the two constituent colors. Screening any color with white produces white. Screening any color with black leaves the original color unchanged.

  The following formula is applied for each of the red, green and blue components:

  $$output_C = overlay_C + input_C - overlay_C * input_C$$

- `overlay` Overlay combines Multiply and Screen blend modes, at half strength. The parts of the overlay layer where the input layer is light become lighter, the parts where the input layer is dark become darker. Areas where the overlay layer are mid grey are unaffected.

  The following formula is applied for each of the red, green and blue components:

  If the input component is less than 0.5 (dark):

  $$output_C = 2 * overlay_C * input_C + input_C * (1 - overlay_A) + overlay_C * (1 - input_A)$$

  If the input component is greater than or equal to 0.5 (light):

  $$output_C = overlay_A * input_A - 2 * (input_A - input_C) * (overlay_A - overlay_C) + input_C * (1 - overlay_A) + overl$$

- `hard-light` The overlay is used to either multiply or screen the input, depending on the overlay color value. If the overlay color value is lighter than 50% gray, the input is screened, otherwise it is multiplied. This is useful for adding highlights or shadows to an image. The implementation is the same of `overlay` but with the roles of input and overlay swapped.

The following formula is applied for each of the red, green and blue components:

If the overlay component is less than 0.5 (dark):

$$output_C = overlay_A * input_A - 2 * (input_A - input_C) * (overlay_A - overlay_C) + input_C * (1 - overlay_A) + overl$$

If the overlay component is greater than or equal to 0.5 (light):

$$output_C = 2 * overlay_C * input_C + input_C * (1 - overlay_A) + overlay_C * (1 - input_A)$$

- `darken` Creates a image that retains the smallest components of the overlay and input pixels. If the overlay pixel has the components r1, g1, and b1, and the input pixel has r2, g2, b2, the resultant pixel is [min(r1,r2), min(g1,g2), min(b1,b2)]

  The following formula is applied for each of the red, green and blue components:

  $$output_C = min(overlay_C, input_C) + input_C * (1 - overlay_A) + overlay_C * (1 - input_A)$$

- `lighten` Lighten has the opposite action of Darken. It selects the maximum of each component from the overlay and input pixels. If the overlay pixel has the components r1, g1, and b1, and the input pixel has r2, g2, b2, the resultant pixel is [max(r1,r2), max(g1,g2), max(b1,b2)]

  The following formula is applied for each of the red, green and blue components:

  $$output_C = max(overlay_C, input_C) + input_C * (1 - overlay_A) + overlay_C * (1 - input_A)$$

- `color-dodge` Divides the input layer by the inverted overlay layer. This lightens the input layer depending on the value of the overlay layer: the brighter the overlay layer, the more its color affects the input layer. Blending any color with white gives white. Blending with black does not change the image.

  The following formula is applied for each of the red, green and blue components:

  If the $overlay_C * input_A$ is greater than or equal to $overlay_A * input_A$ then:

  $$output_C = overlay_A * input_A + input_C * (1 - overlay_A) + overlay_C * (1 - input_A)$$

  If the $overlay_C * input_A$ is less than $overlay_A * input_A$ then:

  $$output_C = input_C * overlay_A/(1 - overlay_C/overlay_A) + overlay_C * (1 - input_A) + input_C * (1 - overlay_A)$$

- `color-burn` Divides the inverted input layer by the overlay layer, and then inverts the result. This darkens the overlay layer increasing the contrast to reflect the color of the input layer. The darker the input layer, the more its color is used.

  The following formula is applied for each of the red, green and blue components:

  If the $overlay_C * input_A$ is greater than or equal to $overlay_A * input_A$ then:

  $$output_C = overlay_A * input_A + input_C * (1 - overlay_A) + overlay_C * (1 - input_A)$$

  If the $overlay_C * input_A$ is greater than $overlay_A * input_A$ then:

  $$output_C = input_C * overlay_A/(1 - overlay_C/overlay_A) + overlay_C * (1 - input_A) + input_C * (1 - overlay_A)$$

Return status code

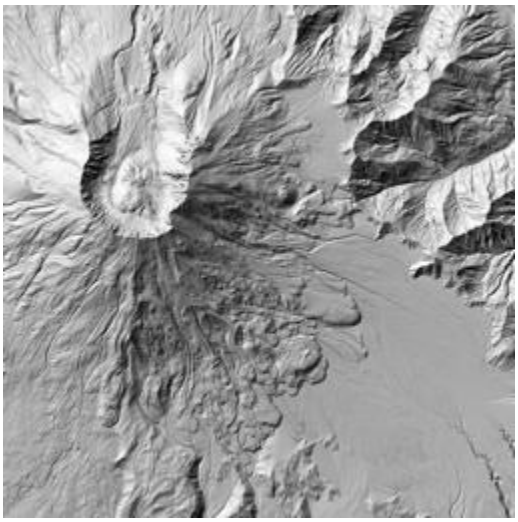
The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

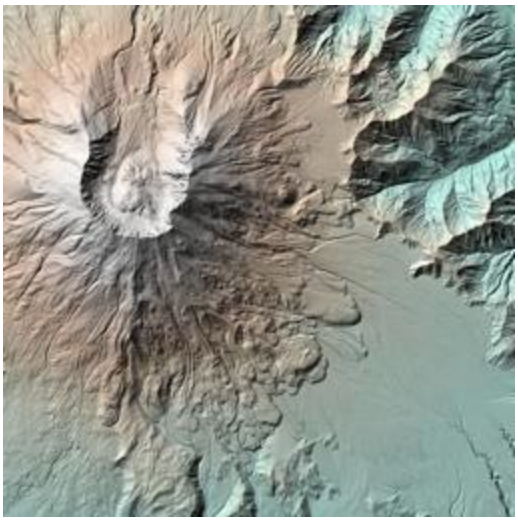
Example 1: Alpha blending of two datasets using 75% opacity for the overlay dataset.

```
$ gdal raster blend --opacity 75 source.tif overlay.tif out.tif
```

Example 2: Combine a hillshade and a hypsometric rendering of a DEM into a colorized hillshade using the hsv-value blending operator.

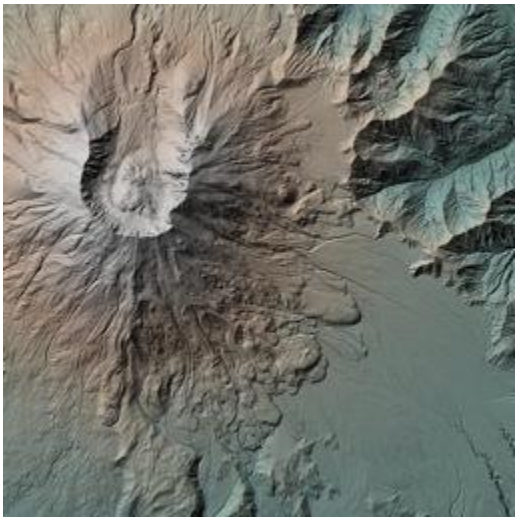


```
$ gdal raster blend --overlay=hillshade.tif --operator=hsv-value \
                    hypsometric.tif hypsometric_combined_with_hillshade.tif
```



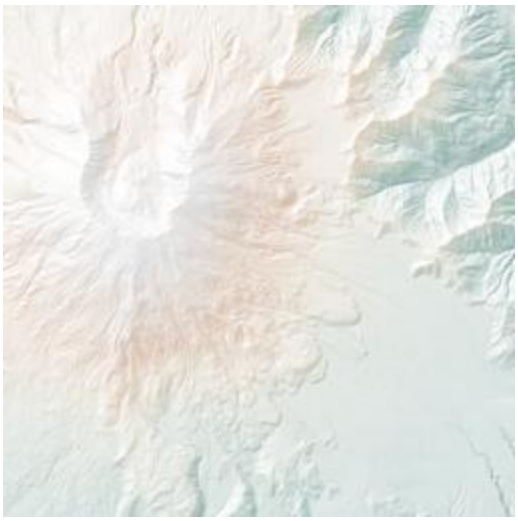
Example 3: Combine a hillshade and a hypsometric rendering of a DEM into a colorized hillshade using the multiply blending operator.

```
$ gdal raster blend --overlay=hillshade.tif --operator=multiply \
                    hypsometric.tif hypsometric_combined_with_hillshade.tif
```



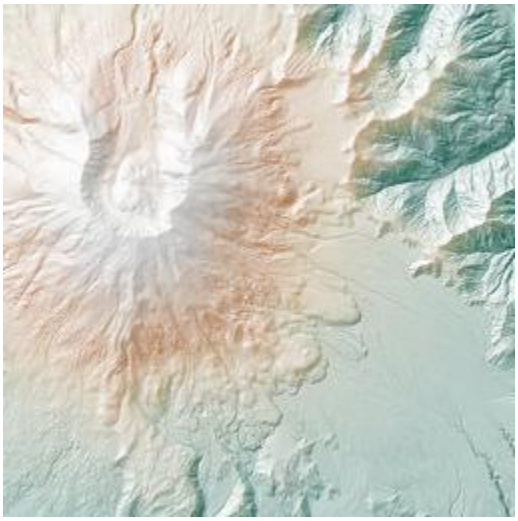
Example 4: Combine a hillshade and a hypsometric rendering of a DEM into a colorized hillshade using the screen blending operator.

```
$ gdal raster blend --overlay=hillshade.tif --operator=screen \
                    hypsometric.tif hypsometric_combined_with_hillshade.tif
```



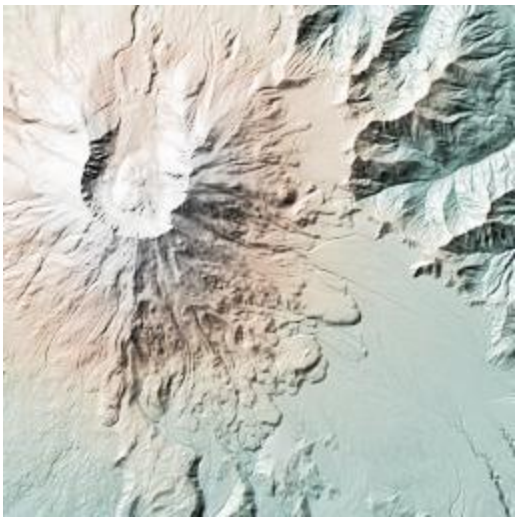
Example 5: Combine a hillshade and a hypsometric rendering of a DEM into a colorized hillshade using the overlay blending operator.

```
$ gdal raster blend --overlay=hillshade.tif --operator=overlay \
                    hypsometric.tif hypsometric_combined_with_hillshade.tif
```



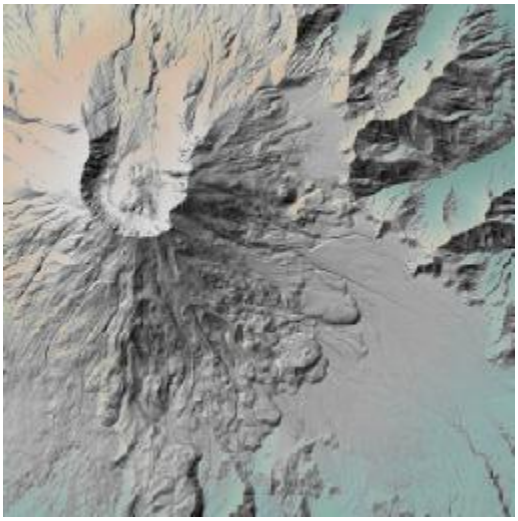
Example 6: Combine a hillshade and a hypsometric rendering of a DEM into a colorized hillshade using the hard-light blending operator.

```
$ gdal raster blend --overlay=hillshade.tif --operator=hard-light \
                    hypsometric.tif hypsometric_combined_with_hillshade.tif
```



Example 7: Combine a hillshade and a hypsometric rendering of a DEM into a colorized hillshade using the darken blending operator.

```
$ gdal raster blend --overlay=hillshade.tif --operator=darken \
                    hypsometric.tif hypsometric_combined_with_hillshade.tif
```



Example 8: Combine a hillshade and a hypsometric rendering of a DEM into a colorized hillshade using the lighten blending operator.

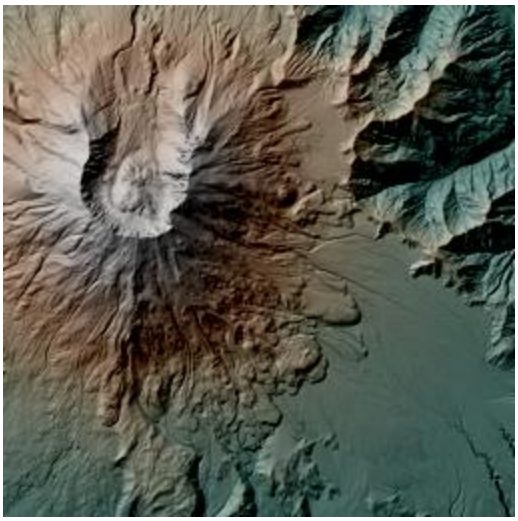
```
$ gdal raster blend --overlay=hillshade.tif --operator=lighten \
                    hypsometric.tif hypsometric_combined_with_hillshade.tif
```

Example 9: Combine a hillshade and a hypsometric rendering of a DEM into a colorized hillshade using the color-dodge blending operator.

```
$ gdal raster blend --overlay=hillshade.tif --operator=color-dodge \
                    hypsometric.tif hypsometric_combined_with_hillshade.tif
```

Example 10: Combine a hillshade and a hypsometric rendering of a DEM into a colorized hillshade using the color-burn blending operator.

```
$ gdal raster blend --overlay=hillshade.tif --operator=color-burn \
                    hypsometric.tif hypsometric_combined_with_hillshade.tif
```



5.1.3.6 gdal raster calc

Added in version 3.11.

Synopsis

```
Usage: gdal raster calc [OPTIONS] <INPUTS>... <OUTPUT>

Perform raster algebra

Positional arguments:
  -i, --input <INPUTS>                                      Input raster datasets [1..␣
↪values] [required] [not available in pipelines]
  -o, --output <OUTPUT>                                     Output raster dataset␣
↪[required] [not available in pipelines]

Common Options:
  -h, --help                                                Display help message and exit
  --json-usage                                              Display usage as JSON␣
↪document and exit
  --config <KEY>=<VALUE>                                    Configuration option [may be␣
↪repeated]
  -q, --quiet                                               Quiet mode (no progress bar␣
↪or warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>       Output format ("GDALG"␣
↪allowed) [not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                     Creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                               Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                            Mutually exclusive with --
↪append
  --append                                                  Append as a subdataset to␣
↪existing output [not available in pipelines]
```

(continues on next page)

(continued from previous page)

```
                                                        Mutually exclusive with --
↪overwrite
  --ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>  Output data type. OUTPUT-DATA-
↪TYPE=UInt8|Int8|UInt16|Int16|UInt32|Int32|UInt64|Int64|CInt16|CInt32|Float16|Float32|Float64|CFloat32
  --no-check-crs                                        Do not check consistency of␣
↪input coordinate reference systems
  --no-check-extent                                     Do not check consistency of␣
↪input extents
  --propagate-nodata                                    Whether to set pixels to the␣
↪output NoData value if any of the input pixels is NoData
  --calc <CALC>                                         Expression(s) to evaluate [1..
↪ values] [required]
  --dialect <DIALECT>                                   Expression dialect.␣
↪DIALECT=muparser|builtin (default: muparser)
  --flatten                                             Generate a single band output␣
↪raster per expression, even if input datasets are multiband
  --nodata <NODATA>                                     Assign a specified nodata␣
↪value to output bands ('none', numeric value, 'nan', 'inf', '-inf')

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                   Input formats [may be␣
↪repeated] [not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                     Open options [may be␣
↪repeated] [not available in pipelines]
```

Description

gdal raster calc performs pixel-wise calculations on one or more input GDAL datasets. Calculations can be performed eagerly, writing results to a conventional raster format, or lazily, written as a set of derived bands in a *VRT (Virtual Dataset)*.

The list of input GDAL datasets can be specified at the end of the command line or put in a text file (one input per line) for very long lists. If more than one input dataset is used, it should be prefixed with a name by which it will be referenced in the calculation, e.g. `A=my_data.tif`. (If a single dataset is used, it will be referred to with the variable `X` in formulas.)

Starting with GDAL 3.12, it is also possible to use a *VRT C++ pixel function* (such a `sum`, `mean`, `min`, `max`) by specifying `--dialect` to `builtin`.

The inputs should have the same spatial reference system and should cover the same spatial extent but are not required to have the same spatial resolution. The spatial extent check can be disabled with `--no-check-extent`, in which case the inputs must have the same dimensions. The spatial reference system check can be disabled with `--no-check-crs`.

Since GDAL 3.12, this algorithm can be part of a *gdal pipeline* or *gdal raster pipeline*.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

--calc

An expression to be evaluated using the muparser math parser library. The expression may refer to individual bands of each input (e.g., `X[1] + 3`) or it may be applied to all bands of an input (`X + 3`).

There are two methods by which an expression may be applied to multiple bands. In the default method, the expression is applied to each band individually, resulting in one output band for each input band. For example, with a three-band input `X`, the expression `--calc "X+3"` would be expanded into `--calc "X[1]+3" --calc "X[2]+3" --calc "X[3]+3"`. If the expression contains a reference to all bands of multiple inputs, those inputs must either have the same number of bands, or a single band. For example, if inputs `A` and `B` each have three bands, and input `C` has a single band, then the argument `--calc "A + B + C"` is equivalent to `--calc "A[1] + B[1] + C[1]" --calc "A[2] + B[2] + C[1]" --calc "A[3] + B[3] + C[1]"`. Similarly, `--calc "sum(A,B,C)"` would be expanded to `--calc "sum(A[1], B[1], C[1]" --calc "sum(A[2], B[2], C[1])" --calc "sum(A[3], B[3], C[1])"`.

In the second method, which is enabled with *`--flatten`*, aggregate functions (`sum`, `avg`, `min`, and `max`) invoked on multiple bands will be applied to all bands in aggregate. In this case, `sum(A)` is expanded into `sum(A[1], A[2], A[3])`, and `sum(A,C)` is expanded into `sum(A[1], A[2], A[3], C[1])`. If the expression consists _only_ of aggregate functions, this will produce a single output band no matter how many input bands are present. However, if the expression contains a non-aggregate reference to all bands of an input, then one output band will still be produced for each input band. A simple example is the expression `A / sum(A)`, which would produce an N-band raster where each output band contains the input band's fraction of the total.

Multiple calculations may be specified; output band(s) will be produced for each expression in the order they are provided.

Input rasters will be converted to 64-bit floating point numbers before performing calculations.

Starting with GDAL 3.12, it is also possible to use a *VRT C++ pixel function* (such a `sum`, `mean`, `min`, `max`) by specifying *`--dialect`* to `builtin`. Arguments to functions are passed within parentheses, like `sum(k=1)`, `min(propagateNoData=true)` or `interpolate_linear(t0=1,dt=1,t=10)`.

Built-in variables

The following built-in variables are available in expressions:

- `_CENTER_X_`: X coordinate of the pixel center, expressed in the dataset CRS
- `_CENTER_Y_`: Y coordinate of the pixel center, expressed in the dataset CRS
- `NODATA`: The output *NoData* value, if any.

These variables are useful for calculations that depend on spatial position, such as latitude-based corrections, solar angle approximations, or zonal masks.

See *Example 7* for a usage example.

Note: To work with longitude/latitude values, the input dataset must be in a geographic coordinate reference system (for example, EPSG:4326).

--dialect `muparser|builtin`

Added in version 3.12.

Specify the expression engine used to interpret the value of *`--calc`*. The default is `muparser`.

--flatten

Added in version 3.12.

Generate a single band output raster per expression, even if input datasets are multiband.

For the default muparser dialect, using the name of an input raster (let's say A) is equivalent to `A[1],A[2],...,A[n]`. Such syntax works with the `min`, `max`, `sum` and `avg` muparser functions.

For example, let's consider 2 rasters A and B with 3bands each, then `--flatten --calc "avg(A)-avg(B)"` is equivalent to `--calc "avg(A[1],A[2],A[3])-avg(B[1],B[2],B[3])"`

And `--flatten --dialect=builtin --calc mean` will compute a single band with the average of all 4 input bands.

-i [<name>=]<input>

Select an input dataset to be processed. If more than one input dataset is provided, each dataset must be prefixed with a name to which it will will be referenced in *--calc*.

--no-check-extent

Do not verify that the input rasters have the same spatial extent. The input rasters will instead be required to have the same dimensions. The geotransform of the first input will be assigned to the output.

--no-check-crs

Do not check the spatial reference systems of the inputs for consistency. All inputs will be assumed to have the spatial reference system of the first input, and this spatial reference system will be used for the output.

--nodata

Added in version 3.12.

Set the NoData value for the output dataset. May be set to "none" to leave the NoData value undefined. If *--nodata* is not specified, **gdal raster calc** will use a NoData value from the first source dataset to have one.

--propagate-nodata

Added in version 3.12.

If set, a NoData value in any input dataset used an in expression will cause the output value to be NoData.

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>

Output data type among `Byte`, `Int8`, `UInt16`, `Int16`, `UInt32`, `Int32`, `UInt64`, `Int64`, `CInt16`, `CInt32`, `Float32`, `Float64`, `CFloat32`, `CFloat64`.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Per-band sum of three files

```
gdal raster calc -i "A=file1.tif" -i "B=file2.tif" -i "C=file3.tif" --calc "A+B+C" -o␣
↪out.tif
```

Example 2: Per-dataset average of all bands of each dataset

```
gdal raster calc -i "A=file1.tif" -i "B=file2.tif" -i "C=file3.tif" --flatten --calc
↪"avg(A)" --calc "avg(B)" --calc "avg(C)" -o out.tif
```

Example 3: Per-band maximum of three files

```
gdal raster calc -i "A=file1.tif" -i "B=file2.tif" -i "C=file3.tif" --calc "max(A,B,C)" -
↪o out.tif
```

Example 4: Setting values of zero and below to NaN

```
gdal raster calc -i "A=input.tif" -o result.tif --calc="A > 0 ? A : NaN"
```

Example 5: Compute the average (as a single band) of all bands of two input datasets

```
gdal raster calc -i A=input1.tif -i B=input2.tif -o result.tif --flatten --calc=mean --
↪dialect=builtin
```

Example 6: Generate a masked aspect layer where the slope angle is greater than 2 degrees, using nested pipelines (since GDAL 3.12.1)

```
gdal raster calc -i "SLOPE=[ read dem.tif ! slope ]" -i "ASPECT=[ read dem.tif ! aspect ]
↪" -o result.tif --calc "(SLOPE >= 2) ? ASPECT : -9999" --nodata -9999
```

Example 7: Latitude-based calculation using _CENTER_Y_

```
gdal raster calc -i A=input.tif \
    --calc="sin(_CENTER_Y_ * 0.0174533)" \
    -o output.tif
```

5.1.3.7 gdal raster clean-collar

Added in version 3.11.

Synopsis

```
Usage: gdal raster clean-collar [OPTIONS] <INPUT> [<OUTPUT>]

Clean the collar of a raster dataset, removing noise.

Positional arguments:
  -i, --input <INPUT>                                    Input raster dataset [required]
  -o, --output <OUTPUT>                                  Output raster dataset

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed
  --update                                               Whether to open existing dataset␣
↪in update mode
  --color <COLOR>                                        Transparent color(s): tuple of␣
↪integer (like 'r,g,b'), 'black', 'white' (default: black) [may be repeated]
  --color-threshold <COLOR-THRESHOLD>                    Select how far from specified␣
↪transparent colors the pixel values are considered transparent. (default: 15)
  --pixel-distance <PIXEL-DISTANCE>                      Number of consecutive transparent␣
↪pixels that can be encountered before the giving up search inwards. (default: 2)
  --add-alpha                                            Adds an alpha band to the output␣
↪dataset.
                                                         Mutually exclusive with --add-mask
  --add-mask                                             Adds a mask band to the output␣
```

(continues on next page)

(continued from previous page)

```
↪dataset.
                                                        Mutually exclusive with --add-
↪alpha
  --algorithm <ALGORITHM>                               Algorithm to apply.␣
↪ALGORITHM=floodfill|twopasses (default: floodfill)

Advanced Options:
  --oo, --open-option <KEY>=<VALUE>                     Open options [may be repeated]
  --if, --input-format <INPUT-FORMAT>                   Input formats [may be repeated]
```

Description

gdal raster clean-collar scans a (8-bit) image and sets all pixels that are nearly or exactly black (the default), white or one or more custom colors around the collar to black or white. This is often used to "fix up" lossy compressed air photos so that color pixels can be treated as transparent when mosaicing. The output format must use lossless compression if either alpha band or mask band is not set.

Program-Specific Options

--add-alpha

Adds an alpha band to the output file. The alpha band is set to 0 in the image collar and to 255 elsewhere. This option is mutually exclusive with *--add-mask*.

--add-mask

Adds a mask band to the output file, The mask band is set to 0 in the image collar and to 255 elsewhere. This option is mutually exclusive with *--add-alpha*.

--algorithm `floodfill|twopasses`

Selects the algorithm to apply.

`twopasses` uses a top-to-bottom pass followed by a bottom-to-top pass. It may miss with concave areas. The algorithm processes the image one scanline at a time. A scan "in" is done from either end setting pixels to black or white until at least "non_black_pixels" pixels that are more than "dist" gray levels away from black, white or custom colors have been encountered at which point the scan stops. The nearly black, white or custom color pixels are set to black or white. The algorithm also scans from top to bottom and from bottom to top to identify indentations in the top or bottom.

`floodfill` (default) uses the Flood Fill algorithm and will work with concave areas. It requires creating a temporary dataset and is slower than `twopasses`. When a non-zero value for *--pixel-distance* is used, `twopasses` is actually called as an initial step of `floodfill`.

--color `<c1>,<c2>,<c3>...<cn>|black|white`

Search for pixels near the specified color. May be specified multiple times. When this option is specified, the pixels that are considered as the collar are set to 0, unless only white is specified, in which case there are set to 255.

--color-threshold `<val>`

Select how far from black, white or custom colors the pixel values can be and still considered near black, white or custom color. Defaults to 15.

--pixel-distance `<val>`

Number of consecutive transparent pixels that can be encountered before the giving up search inwards. Said otherwise, the collar will be extended by this number of pixels. Defaults to 2.

Standard Options

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep + | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--update

If only an input dataset is specified, ask for it to be opened in update mode If an output dataset is specified, ask for it to be opened in update mode (this implies that it already exists). Note that updating an existing dataset may lead to file size increase if the dataset is compressed, and/or quality loss if lossy compression is used.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Edit in place a dataset using the default black color as the color of the collar

```
$ gdal raster clean-collar --update my.tif
```

Example 2: Create a new dataset, using black or white as the color of the collar, considering values in the [0,5] range as being considered black, and values in the [250,255] range to be white. It also adds an alpha band to the output dataset.

```
$ gdal raster clean-collar --add-alpha --color=black --color=white --color-threshold=5␣
↪in.tif out.tif
```

5.1.3.8 gdal raster clip

Added in version 3.11.

Synopsis

```
Usage: gdal raster clip [OPTIONS] <INPUT> <OUTPUT>

Clip a raster dataset.
```

(continues on next page)

(continued from previous page)

```
Positional arguments:
  -i, --input <INPUT>                                   Input raster datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                 Output raster dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                            Display help message and exit
  --json-usage                                          Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                Configuration option [may be␣
↪repeated]
  -q, --quiet                                           Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                 Creation option [may be repeated]␣
↪[not available in pipelines]
  --overwrite                                           Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                        Mutually exclusive with --append
  --append                                              Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                        Mutually exclusive with --
↪overwrite
  --bbox <BBOX>                                         Clipping bounding box as xmin,
↪ymin,xmax,ymax
                                                        Mutually exclusive with --window,␣
↪--geometry, --like
  --bbox-crs <BBOX-CRS>                                 CRS of clipping bounding box
  --window <WINDOW>                                     Raster window as col,line,width,
↪height in pixels
                                                        Mutually exclusive with --bbox, --
↪geometry, --like
  --geometry <GEOMETRY>                                 Clipping geometry (WKT or GeoJSON)
                                                        Mutually exclusive with --bbox, --
↪window, --like
  --geometry-crs <GEOMETRY-CRS>                         CRS of clipping geometry
  --like <DATASET>                                      Dataset to use as a template for␣
↪bounds
                                                        Mutually exclusive with --bbox, --
↪window, --geometry
  --like-sql <SELECT-STATEMENT>                         SELECT statement to run on the
↪'like' dataset
                                                        Mutually exclusive with --like-
↪where
  --like-layer <LAYER-NAME>                             Name of the layer of the 'like'␣
↪dataset
  --like-where <WHERE-EXPRESSION>                       WHERE SQL clause to run on the
```

(continues on next page)

(continued from previous page)

```
↪'like' dataset
                                                        Mutually exclusive with --like-sql
  --only-bbox                                           For 'geometry' and 'like', only␣
↪consider their bounding box
  --allow-bbox-outside-source                           Allow clipping box to include␣
↪pixels outside input dataset
  --add-alpha                                           Adds an alpha mask band to the␣
↪destination when the source raster have none.

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                   Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                     Open options [may be repeated]␣
↪[not available in pipelines]
```

Description

gdal raster clip can be used to clip a raster dataset using georeferenced coordinates.

Either *--bbox* or *--like* must be specified.

The output dataset is in the same SRS as the input one, and the original resolution is preserved. Bounds are rounded to match whole pixel locations (i.e. there is no resampling involved)

`clip` can also be used as a step of *gdal raster pipeline*.

Program-Specific Options

--add-alpha

Adds an alpha mask band to the destination when the source raster has none.

--allow-bbox-outside-source

If set, allows the bounds indicated by *--bbox* to cover an extent that is greater than the input dataset. Output pixels from areas beyond the input extent will be set to zero or the NoData value of the input dataset.

--bbox <xmin>,<ymin>,<xmax>,<ymax>

Bounds to which to clip the dataset. They are assumed to be in the CRS of the input dataset, unless *--bbox-crs* is specified. The X and Y axis are the "GIS friendly ones", that is X is longitude or easting, and Y is latitude or northing. The bounds are expanded if necessary to match input pixel boundaries. By default, **gdal raster clip** will produce an error if the bounds indicated by *--bbox* are greater than the extents of input dataset. This check can be bypassed using *--allow-bbox-outside-source*.

--bbox-crs <CRS>

CRS in which the <xmin>,<ymin>,<xmax>,<ymax> values of *--bbox* are expressed. If not specified, it is assumed to be the CRS of the input dataset. Note that specifying *--bbox-crs* does not cause the raster to be reprojected. Instead, the bounds are reprojected from the bbox-crs to the CRS of the input dataset.

--geometry <WKT_or_GeoJSON>

Geometry as a WKT or GeoJSON string of a polygon (or multipolygon) to which to clip the dataset. Raster areas within the bounding box of the geometry but not inside the geometry itself will be set to the nodata value of the raster, or 0 if there is none. All pixels overlapping the geometry will be selected. If the input geometry is GeoJSON, its CRS is assumed to be WGS84, unless there is a CRS defined in the GeoJSON geometry or *--geometry-crs* is specified. If the input geometry is WKT, its CRS is assumed to be the one of the input dataset, unless *--geometry-crs* is specified. The X and Y axis are the "GIS friendly ones", that is X is longitude or easting, and Y is latitude or northing. Mutually exclusive with *--bbox* and *--like*.

--geometry-crs <CRS>

CRS in which the coordinates values of *--geometry* are expressed. If not specified, it is assumed to be the CRS of the input dataset. The bounds are reprojected from the geometry-crs to the CRS of the input dataset.

--like <DATASET>

Vector or raster dataset to use as a template for bounds. If the specified dataset is a raster, its rectangular bounds are used as the clipping geometry. If the specified dataset is a vector dataset, its polygonal geometries are unioned together to form the clipping geometry. If several layers are present, *--like-sql* or *--like-layer* must be specified. Raster areas within the bounding box of the geometry but not inside the geometry itself will be set to the nodata value of the raster, or 0 if there is none. Mutually exclusive with *--bbox* and *--geometry*.

--like-layer <LAYER-NAME>

Select the named layer from the vector clip dataset. Mutually exclusive with *--like-sql*

--like-sql <SELECT-STATEMENT>

Select desired geometries from the vector clip dataset using an SQL query. e.g `SELECT geom FROM my_layer WHERE country = 'France'`. The SQL dialect used will be the default one of the `like` dataset (OGR SQL for Shapefile, SQLite for GeoPackage, PostgreSQL for PostGIS, etc.). Mutually exclusive with *--like-layer* and *--like-where*

--like-where <WHERE-EXPRESSION>

Restrict desired geometries from vector clip dataset layer based on an attribute query. e.g `country = 'France'`.

--only-bbox

For *--geometry* and *--like*, only consider the bounding box of the geometry.

--window <column>,<line>,<width>,<height>

Added in version 3.12.

Selects a subwindow from the source image for copying based on pixel/line location. Pixel/line offsets (`column` and `line`) are measured from the left and top of the image. If the subwindow extends beyond the bounds of the source dataset, output pixels will be written with a value of zero, unless a NoData value is defined either the source dataset.

Standard Options

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Clip a GeoTIFF file to the bounding box from longitude 2, latitude 49, to longitude 3, latitude 50 in WGS 84

```
$ gdal raster clip --bbox=2,49,3,50 --bbox-crs=EPSG:4326 in.tif out.tif --overwrite
```

Example 2: Clip a GeoTIFF file using the bounds of reference.tif

```
$ gdal raster clip --like=reference.tif in.tif out.tif --overwrite
```

Example 3: Clip a GeoTIFF file from raster column 1000 and line 2000, for a width of 500 pixels and a height of 600 pixels

```
$ gdal raster clip --window=1000,2000,500,600 in.tif out.tif --overwrite
```

Example 4: Clip a GeoTIFF file using vector features

```
$ gdal raster clip NE2_50M_SR_W.tif clipped.tif \
     --like natural_earth_vector.gpkg \
     --like-layer "ne_50m_admin_0_countries" \
     --like-where "ADMIN='Romania'"
```

Example 5: Clip a GeoTIFF file using SQL

This example produces the same result as *Example 4*, but uses *--like-sql* instead of *--like-layer* and *--like-where*.

The geometry column name (`geom` in this case) must be included in the SQL query.

```
$ gdal raster clip NE2_50M_SR_W.tif clipped.tif \
    --like natural_earth_vector.gpkg \
    --like-sql "SELECT geom FROM ne_50m_admin_0_countries WHERE ADMIN = 'Romania'" \
    --overwrite
```

Example 6: Clip a GeoTIFF using a bounding box extending beyond the source extent, and write the result as a COG

Use `--allow-bbox-outside-source` to suppress the error `ERROR 1: Computed source window -180 2943 6269 6235 falls partially outside source raster extent` when the bounding box exceeds the raster extent.

```
$ gdal raster clip \
    --bbox=3757032.814272985,-626172.1357121654,4383204.9499851465,0 \
    --bbox-crs=EPSG:3857 \
    --allow-bbox-outside-source \
    in.tif out.tif \
    --overwrite \
    --output-format COG
```

5.1.3.9 gdal raster color-map

Added in version 3.11.

Synopsis

```
Usage: gdal raster color-map [OPTIONS] <INPUT> <OUTPUT>

Generate a RGB or RGBA dataset from a single band, using a color map

Positional arguments:
  -i, --input <INPUT>                                    Input raster datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output raster dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                         Mutually exclusive with --append
  --append                                               Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                         Mutually exclusive with --
↪overwrite
  -b, --band <BAND>                                      Input band (1-based index)␣
```

(continues on next page)

(continued from previous page)

```
↪(default: 1)
  --color-map <COLOR-MAP>                                Color map filename
  --add-alpha                                            Adds an alpha mask band to the␣
↪destination.
  --color-selection <COLOR-SELECTION>                    How to compute output colors from␣
↪input values. COLOR-SELECTION=interpolate|exact|nearest (default: interpolate)

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                    Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                      Open options [may be repeated]␣
↪[not available in pipelines]
```

Description

gdal raster color-map generates a RGB or RGBA dataset from a single band, using a color map, either attached directly to a raster band, or from an external text file. It is typically used to create hypsometric maps from a DEM.

This subcommand is also available as a potential step of *gdal raster pipeline*

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

--add-alpha

Adds an alpha mask band to the output.

-b, --band <BAND>

Index (starting at 1) of the band to process.

--color-map <COLOR-MAP>

Color map filename containing the association between various raster values and the corresponding wished color.

This option must be specified if the input raster band has no attached color table.

The text-based color configuration file generally contains 4 columns per line: the raster value and the corresponding Red, Green, Blue component (between 0 and 255). The raster value can be any floating point value, or the `nv` keyword for the nodata value. The raster can also be expressed as a percentage: 0% being the minimum value found in the raster, 100% the maximum value.

An extra column can be optionally added for the alpha component. If it is not specified, full opacity (255) is assumed.

Various field separators are accepted: comma, tabulation, spaces, ':'.

Common colors used by GRASS can also be specified by using their name, instead of the RGB triplet. The supported list is: white, black, red, green, blue, yellow, magenta, cyan, aqua, grey/gray, orange, brown, purple/violet and indigo.

GMT `.cpt` palette files are also supported (COLOR_MODEL = RGB only).

Note: the syntax of the color configuration file is derived from the one supported by GRASS r.colors utility. ESRI HDR color table files (.clr) also match that syntax. The alpha component and the support of tab and comma as separators are GDAL specific extensions.

For example:

```
3500   white
2500   235:220:175
50%   190 185 135
700    240 250 150
0      50  180  50
nv     0   0   0   0
```

To implement a "round to the floor value" mode, the raster value can be duplicate with a new value being slightly above the threshold. For example to have red in [0,10], green in]10,20] and blue in]20,30]:

```
0       red
10      red
10.001  green
20      green
20.001  blue
30      blue
```

--color-selection `interpolate|exact|nearest`

How to compute output colors from input values:

- `interpolate` (default): apply linear interpolation when a raster value falls between two entries of the color map.
- `exact`: only input raster values matching exacting an entry of the color map will be colorized. If none matching color entry is found, the "0,0,0,0" RGBA quadruplet will be used.
- `nearest`: input raster values will be assigned the entry of the color map that is the closest.

> **Note**
>
> This option is only taken into account when *--color-map* is specified.

Standard Options

--co, --creation-option `<NAME>=<VALUE>`

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

-f, --of, --format, --output-format `<OUTPUT-FORMAT>`

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Generates a RGB dataset from a DTED0 file using an external color map

```
$ gdal raster color-map --color-map=color-map.txt n43.dt0 out.tif --overwrite
```

5.1.3.10 gdal raster contour

Added in version 3.11.

Synopsis

```
Usage: gdal raster contour [OPTIONS] <INPUT> <OUTPUT>

Creates a vector contour from a raster elevation model (DEM).

Positional arguments:
  -i, --input <INPUT>                                   Input raster datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                 Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                            Display help message and exit
  --json-usage                                          Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                Configuration option [may be␣
↪repeated]
  -q, --quiet                                           Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                 Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>          Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                           Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --output-layer <OUTPUT-LAYER>                         Output layer name
  -b, --band <BAND>                                     Input band (1-based index)␣
↪(default: 1)
```

(continues on next page)

(continued from previous page)

```
  --elevation-name <ELEVATION-NAME>                    Name of the elevation field
  --min-name <MIN-NAME>                                Name of the minimum elevation␣
↪field
  --max-name <MAX-NAME>                                Name of the maximum elevation␣
↪field
  --3d                                                 Force production of 3D vectors␣
↪instead of 2D
  --input-nodata <INPUT-NODATA>                        Input pixel value to treat as
↪'nodata'
  --interval <INTERVAL>                                Elevation interval between␣
↪contours
                                                       Mutually exclusive with --levels,␣
↪--exp-base
  --levels <LEVELS>                                    List of contour levels [may be␣
↪repeated]
                                                       Mutually exclusive with --
↪interval, --exp-base
  -e, --exp-base <EXP-BASE>                            Base for exponential contour␣
↪level generation
                                                       Mutually exclusive with --
↪interval, --levels
  --off, --offset <OFFSET>                             Offset to apply to contour levels
  -p, --polygonize                                     Create polygons instead of lines
  --group-transactions <GROUP-TRANSACTIONS>            Group n features per transaction␣
↪(default 100 000)

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                  Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                    Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>      Output open options [may be␣
↪repeated] [not available in pipelines]
```

Description

gdal raster contour creates a vector contour from a raster elevation model (DEM).

Since GDAL 3.12, this algorithm can be part of a *gdal pipeline*.

Added in version 3.12.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

> **Note**
>
> However this algorithm is not natively streaming compatible. Consequently a in-memory temporary dataset will be generated, which may cause significant processing time at opening.

Program-Specific Options

--3d

Forces the production of 3D vectors instead of 2D. Includes elevation at every vertex.

-b, --band <BAND>

Picks a particular band to get the DEM from. Defaults to band 1.

--elevation-name <ELEVATION-NAME>

Provides a name for the attribute in which to put the elevation. If not provided no elevation attribute is attached. Ignored in polygonal contouring (-p) mode.

-e, --exp-base <EXP-BASE>

Generate levels on an exponential scale: base ^ k, for k an integer. Mutually exclusive with *--interval*, *--levels*.

--group-transactions <GROUP-TRANSACTIONS>

Group n features per transaction (default 100 000).

--interval <INTERVAL>

Elevation interval between contours. Mutually exclusive with *--levels*, *--exp-base*. The first contour will be generated at the first multiple of INTERVAL which is greater than the raster minimum value.

--levels <LEVELS>

List of contour levels. *MIN* and *MAX* are special values that represent the minimum and maximum values in the raster. When *--polygonize* is used, the specified values are used as bounds of the generated polygons. Mutually exclusive with *--interval*, *--exp-base*.

--max-name <MAX-NAME>

Provides a name for the attribute in which to put the maximum elevation. If not provided no maximum elevation attribute is attached. Ignored in polygonal contouring (-p) mode.

--min-name <MIN-NAME>

Provides a name for the attribute in which to put the minimum elevation. If not provided no minimum elevation attribute is attached. Ignored in polygonal contouring (-p) mode.

--off, --offset <OFFSET>

Offset from zero relative to which to interpret intervals.

--nln, --output-layer <OUTPUT-LAYER>

Provides a name for the output vector layer. Defaults to "contour".

-p, --polygonize

Create polygons instead of lines.

--input-nodata <INPUT-NODATA>

Input pixel value to treat as 'nodata'.

Standard Options

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Create contour lines from a raster elevation model

The following example creates contour lines from a raster elevation model:

```
gdal raster contour --interval 100 elevation.tif contour.shp
```

This will create a shapefile named `contour.shp` with contour lines at 100 meter intervals.

Example 2: Create contour polygons from a raster elevation model with custom attributes and fixed levels

The following example creates contour polygons from a raster elevation model with custom attributes and fixed levels:

```
gdal raster contour --levels MIN,100,200,MAX --polygonize --min-name MIN --max-
↪name MAX elevation.tif contour.shp
```

This will create a shapefile named `contour.shp` with contour polygons from the minimum raster value to 100, and from 100 to 200 and from 200 to the maximum value, and with attributes `MIN`, and `MAX`.

5.1.3.11 gdal raster compare

Added in version 3.12.

Synopsis

```
Usage: gdal raster compare [OPTIONS] <REFERENCE> <INPUT>

Compare two raster datasets.

Positional arguments:
  --reference <REFERENCE>              Reference dataset [required]
  -i, --input <INPUT>                  Input raster dataset [required] [not available in␣
↪pipelines]

Common Options:
  -h, --help                           Display help message and exit
  --json-usage                         Display usage as JSON document and exit
  --config <KEY>=<VALUE>               Configuration option [may be repeated]
  -q, --quiet                          Quiet mode (no progress bar or warning message)␣
↪[not available in pipelines]

Options:
  --skip-all-optional                  Skip all optional comparisons
  --skip-binary                        Skip binary file comparison
  --skip-crs                           Skip CRS comparison
  --skip-geotransform                  Skip geotransform comparison
  --skip-overview                      Skip overview comparison
  --skip-metadata                      Skip metadata comparison
  --skip-rpc                           Skip RPC metadata comparison
  --skip-geolocation                   Skip Geolocation metadata comparison
  --skip-subdataset                    Skip subdataset comparison

Advanced Options:
  --if, --input-format <INPUT-FORMAT>  Input formats [may be repeated] [not available in␣
↪pipelines]
```

(continues on next page)

(continued from previous page)

```
  --oo, --open-option <KEY>=<VALUE>    Open options [may be repeated] [not available in␣
↪pipelines]
```

Description

gdal raster compare compares two GDAL supported datasets and reports the differences. In addition to reporting differences to the standard output, the program will also return the difference count in its exit value.

As a convention, the first dataset specified as a positional argument, or through *--reference*, is assumed to be the reference/exact/golden dataset. The second dataset specified as a positional argument, or through *--input*, is the dataset compared to the reference dataset.

Image pixels, and various metadata are checked. There is also a byte by byte comparison done which will count as one difference. So, if it is only important that the GDAL visible data is identical, a difference count of 1 (the binary difference) should be considered acceptable, or you may specify *--skip-binary* to omit byte to byte comparison.

This program can also be used as the last step of a *raster pipeline*.

The following options are available:

Program-Specific Options

--input <input-dataset>

The dataset being compared to the reference dataset, referred to as the input dataset.

--reference <reference-dataset>

The dataset that is considered correct, referred to as the reference dataset.

--skip-all-optional

Whether to skip all optional tests. This is an alias to defining all other `--skip-XXXX` options.

The remaining non-optional tests are:

- checking dataset width, height and band count
- checking band description, data type, color interpretation and nodata value
- checking pixel content

--skip-binary

Whether to skip exact comparison of binary content.

--skip-crs

Whether to skip comparison of coordinate reference systems (CRS).

--skip-geolocation

Whether to skip comparison of GEOLOCATION metadata domain.

--skip-geotransform

Whether to skip comparison of geotransform matrix.

--skip-metadata

Whether to skip comparison of metadata

--skip-overview

Whether to skip comparison of overviews.

--skip-rpc

Whether to skip comparison of Rational Polynomial Coefficients (RPC) metadata domain.

--skip-subdataset

Whether to ignore comparison of all subdatasets that are part of the dataset.

Standard Options

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Comparing two datasets that differ by their data types

```
$ gdal raster compare autotest/gcore/data/byte.tif autotest/gcore/data/uint16.tif
Reference file has size 736 bytes, whereas input file has size 1136 bytes.
Reference band 1 has data type Byte, but input band has data type UInt16

$ echo $?
2
```

Example 2: Comparing two datasets while values have different units

```
$ gdal raster pipeline read test_in_foot.tif ! calc --calc "X * 0.3048" ! compare --
↪reference=reference_in_meter.tif
```

5.1.3.12 gdal raster convert

Added in version 3.11.

Synopsis

```
Usage: gdal raster convert [OPTIONS] <INPUT> <OUTPUT>

Convert a raster dataset.

Positional arguments:
```

(continues on next page)

(continued from previous page)

```
  -i, --input <INPUT>                                    Input raster datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output raster dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                         Mutually exclusive with --append
  --append                                               Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                         Mutually exclusive with --
↪overwrite

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                    Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                      Open options [may be repeated]␣
↪[not available in pipelines]
```

Description

gdal raster convert can be used to convert raster data between different formats.

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation

for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

-q, --quiet

Suppress progress bar and some warning messages.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Converting file utm.tif to a cloud-optimized GeoTIFF using JPEG compression

```
$ gdal raster convert --format=COG --co COMPRESS=JPEG utm.tif utm_cog.tif
```

Example 2: Converting file utm.tif to GeoPackage raster

```
$ gdal raster convert utm.tif utm.gpkg
```

5.1.3.13 gdal raster create

Added in version 3.11.

Synopsis

```
Usage: gdal raster create [OPTIONS] <OUTPUT>

Create a new raster dataset.

Positional arguments:
  -o, --output <OUTPUT>                                    Output raster dataset␣
```

(continues on next page)

(continued from previous page)

```
↪[required] [not available in pipelines]

Common Options:
  -h, --help                                            Display help message and exit
  --json-usage                                          Display usage as JSON␣
↪document and exit
  --config <KEY>=<VALUE>                                Configuration option [may be␣
↪repeated]
  -q, --quiet                                           Quiet mode (no progress bar␣
↪or warning message) [not available in pipelines]

Options:
  -i, --like, --input <TEMPLATE-DATASET>                Template raster dataset [not␣
↪available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>   Output format ("GDALG"␣
↪allowed) [not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                 Creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                           Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                        Mutually exclusive with --
↪append
  --append                                              Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                        Mutually exclusive with --
↪overwrite
  --size <width>,<height>                               Output size in pixels
  --band-count <BAND-COUNT>                             Number of bands (default: 1)
  --ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>  Output data type. OUTPUT-DATA-
↪TYPE=UInt8|Int8|UInt16|Int16|UInt32|Int32|UInt64|Int64|CInt16|CInt32|Float16|Float32|Float64|CFloat32
↪(default: Byte)
  --nodata <NODATA>                                     Assign a specified nodata␣
↪value to output bands ('none', numeric value, 'nan', 'inf', '-inf')
  --burn <BURN>                                         Burn value [may be repeated]
  --crs <CRS>                                           Set CRS
  --bbox <BBOX>                                         Bounding box as xmin,ymin,
↪xmax,ymax
  --metadata <KEY>=<VALUE>                              Add metadata item [may be␣
↪repeated]
  --copy-metadata                                       Copy metadata from input␣
↪dataset
                                                        Depends on --input
  --copy-overviews                                      Create same overview levels␣
↪as input dataset
                                                        Depends on --input

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                   Input formats [may be␣
↪repeated] [not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                     Open options [may be␣
↪repeated] [not available in pipelines]
```

Description

gdal raster create can be used to initialize a new raster file, from its dimensions, band count, CRS, geotransform, nodata value and metadata.

The new file can also be initialized from a model input file with the optional *--like* option, copying its properties but not its pixel values. By default, metadata and the overview structure are not copied from the model input file, unless *--copy-metadata* and *--copy-overviews* are specified. Options *--size*, *--band-count*, *--datatype*, *--nodata*, *--crs*, *--bbox*, *--metadata* can be used to override the values inherited from the model input file.

For GeoTIFF output, setting the `SPARSE_OK` creation option to `YES` can be useful to create a file of minimum size.`

gdal raster create can be used also in special cases, like creating a PDF file from a XML composition file.

Since GDAL 3.13, `create` can also be used as a step of *gdal raster pipeline*.

Program-Specific Options

--band-count <count>

Number of bands. Defaults to 1.

--bbox <xmin>,<ymin>,<xmax>,ymax>

Sets the spatial bounding box, in CRS units. 'x' is longitude values for geographic CRS and easting for projected CRS. 'y' is latitude values for geographic CRS and northing for projected CRS.

--burn <value>

A fixed value to burn into a band. A list of *--burn* options can be supplied, one per band (the first value will apply to the first band, the second one to the second band, etc.). If a single value is specified, it will apply to all bands.

--copy-metadata

Copy metadata from input dataset and raster bands. Requires *--like* to be specified.

--copy-overviews

Create same overview levels as input dataset (but with empty content). Requires *--like* to be specified.

--crs <CRS>

Set CRS.

The coordinate reference systems that can be passed are anything supported by the *OGRSpatialReference::SetFromUserInput()* call, which includes EPSG Projected, Geographic or Compound CRS (i.e. EPSG:4296), a well known text (WKT) CRS definition, PROJ.4 declarations, or the name of a .prj file containing a WKT CRS definition.

`null` or `none` can be specified to unset the existing CRS of the *--like* dataset if it is set.

Note that the spatial extent is also left unchanged.

-i, --like, --input <TEMPLATE-DATASET>

Name of GDAL input dataset that serves as a template for default values of options *--size*, *--band-count*, *--datatype*, *--crs*, *--bbox* and *--nodata*. Note that the pixel values will *not* be copied.

Since GDAL 3.13, if the input dataset is tiled, and for output formats `GTiff`, `COG` and `GPKG`, its tile dimensions are replicated to the output file, when they are compatible with its capabilities, and if the user hasn't specified any creation option related to tiling.

--metadata <KEY>=<VALUE>

Adds a metadata item, at the dataset level.

--nodata <value>

Sets the nodata value.

null or none can be specified to unset the existing nodata value of the *--like* dataset if it is set. nan, inf or -inf are also accepted for floating point rasters to respectively mean the special values not-a-number, positive infinity and minus infinity.

--size <xsize>,<ysize>

Set the size of the output file in pixels. First value is width. Second one is height.

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

--ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>

Output data type among Byte, Int8, UInt16, Int16, UInt32, Int32, UInt64, Int64, CInt16, CInt32, Float32, Float64, CFloat32, CFloat64.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by gdal --formats | grep raster | grep + | sort

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Initialize a new GeoTIFF file with 3 bands and a uniform value of 10

```
gdal raster create --size=20,20 --band-count=3 --crs=EPSG:4326 --bbox=2,49,3,50 --burn␣
↪10 out.tif
```

Example 2: Create a PDF file from a XML composition file

```
gdal raster create --creation-option COMPOSITION_FILE=composition.xml out.pdf
```

Example 3: Initialize a blank GeoTIFF file from an input one

```
gdal raster create --like prototype.tif output.tif
```

5.1.3.14 gdal raster edit

Added in version 3.11.

Synopsis

```
Usage: gdal raster edit [OPTIONS] <DATASET>

Edit a raster dataset.

Positional arguments:
  --dataset <DATASET>                                       Dataset (to be updated in-
↪place, unless --auxiliary) [required] [not available in pipelines]

Common Options:
  -h, --help                                                Display help message and␣
↪exit
  --json-usage                                              Display usage as JSON␣
↪document and exit
  --config <KEY>=<VALUE>                                    Configuration option [may␣
↪be repeated]
  -q, --quiet                                               Quiet mode (no progress␣
↪bar or warning message) [not available in pipelines]

Options:
  --auxiliary                                               Ask for an auxiliary .aux.
↪xml file to be edited [not available in pipelines]
  -b, --band <BAND>                                         Active band (1-based index)
  --crs <CRS>                                               Override CRS (without␣
↪reprojection)
  --bbox <BBOX>                                             Bounding box as xmin,ymin,
```

(continues on next page)

(continued from previous page)

```
↪xmax,ymax
  --nodata <NODATA>                                          Assign a specified nodata␣
↪value to output bands ('none', numeric value, 'nan', 'inf', '-inf')
  --color-interpretation <[all|<BAND>=]<COLOR-INTEPRETATION>  Set band color␣
↪interpretation [may be repeated]
  --color-map <COLOR-MAP>                                    Color map filename
                                                             Mutually exclusive with --
↪unset-color-table
  --scale <[<BAND>=]<SCALE>                                  Override band scale factor␣
↪[may be repeated]
  --offset <[<BAND>=]<OFFSET>                                Override band offset␣
↪constant [may be repeated]
  --metadata <KEY>=<VALUE>                                   Add/update dataset␣
↪metadata item [may be repeated]
  --unset-color-table                                        Unset color table on 1st␣
↪band or the one specified with --band
                                                             Mutually exclusive with --
↪color-map
  --unset-metadata <KEY>                                     Remove dataset metadata␣
↪item(s) [may be repeated]
  --unset-metadata-domain <DOMAIN>                           Remove dataset metadata␣
↪domain(s) [may be repeated]
  --gcp <GCP>                                                Add ground control point,␣
↪formatted as pixel,line,easting,northing[,elevation], or @filename [may be repeated]
  --stats                                                    Compute statistics, using␣
↪all pixels
                                                             Mutually exclusive with --
↪approx-stats
  --approx-stats                                             Compute statistics, using␣
↪a subset of pixels
                                                             Mutually exclusive with --
↪stats
  --hist                                                     Compute histogram

Advanced Options:
  --oo, --open-option <KEY>=<VALUE>                          Open options [may be␣
↪repeated] [not available in pipelines]
```

Description

gdal raster edit can be used to edit a raster dataset.

This subcommand is also available as a potential step of *gdal raster pipeline*

Program-Specific Options

--approx-stats

Compute raster band statistics for all bands. They may be computed based on overviews or a subset of all tiles. Useful if you are in a hurry and don't need precise stats.

> **Note**
>
> This option is not available when the command is part of a pipeline.

--auxiliary

Force opening the dataset in read-only mode. For drivers that implement the Persistent Auxiliary Metadata (PAM) mechanism, changes will be saved in an auxiliary side car file of extension `.aux.xml`.

-b, --band <BAND>

Added in version 3.13.

Index (starting at 1) of the band to which the following options apply: *--color-map*, *--unset-color-table*. Defaults to one if not specified.

--bbox <xmin>,<ymin>,<xmax>,ymax>

Override the spatial bounding box, in CRS units, without reprojecting or subsetting. 'x' is longitude values for geographic CRS and easting for projected CRS. 'y' is latitude values for geographic CRS and northing for projected CRS.

--color-map <COLOR-MAP>

Added in version 3.13.

Color map filename containing the association between various raster values and the corresponding wished color to set on the first band, or the one specified by *--band*. This can be either a GDAL dataset with a color table attached to its first band or a text-based color configuration

The text-based color configuration file generally contains 4 columns per line: the raster value and the corresponding Red, Green, Blue component (between 0 and 255). The raster value mus be an integer value between 0 and 65535. The raster value can also be expressed as a percentage: 0% being the minimum value found in the raster, 100% the maximum value.

An extra column can be optionally added for the alpha component. If it is not specified, full opacity (255) is assumed.

Various field separators are accepted: comma, tabulation, spaces, ':'.

Common colors used by GRASS can also be specified by using their name, instead of the RGB triplet. The supported list is: white, black, red, green, blue, yellow, magenta, cyan, aqua, grey/gray, orange, brown, purple/violet and indigo.

GMT `.cpt` palette files are also supported (COLOR_MODEL = RGB only).

Note: the syntax of the color configuration file is derived from the one supported by GRASS r.colors utility. ESRI HDR color table files (.clr) also match that syntax. The alpha component and the support of tab and comma as separators are GDAL specific extensions.

For example:

```
3500   white
2500   235:220:175
50%   190 185 135
700    240 250 150
0      50  180  50
```

--crs <CRS>

Override CRS, without reprojecting.

The coordinate reference systems that can be passed are anything supported by the *OGRSpatialReference::SetFromUserInput()* call, which includes EPSG Projected, Geographic or

Compound CRS (i.e. EPSG:4296), a well known text (WKT) CRS definition, PROJ.4 declarations, or the name of a .prj file containing a WKT CRS definition.

`null` or `none` can be specified to unset an existing CRS.

Note that the spatial extent is also left unchanged.

--dataset <DATASET>

Dataset name, to be updated in-place, unless *--auxiliary* is set. Required.

--gcp <GCP>

Added in version 3.12.

Set ground control point(s), replacing any existing GCPs. Each GCP must be formatted as a string "pixel,line,easting,northing" or "pixel,line,easting,northing,elevation". Each GCP must be specified with a `--gcp` argument.

It is also possible to provide a single `--gcp` argument whose value is the filename of a vector dataset, prefixed with `@`. This dataset must have a single layer with the following required fields `column`, `line`, `x`, `y`, and optionally `id`, `info` and `z`.

--hist

Compute histogram information for all bands.

> **Note**
>
> This option is not available when the command is part of a pipeline.

--color-interpretation [all|<BAND>=]<COLOR-INTERPRETATION>

Added in version 3.13.

Set band color interpretation. May be repeated.

<COLOR-INTERPRETATION> is one of `red`, `green`, `blue`, `alpha`, `gray`, `undefined`, `pan`, `coastal`, `rededge`, `nir`, `swir`, `mwir` or `lwir`.

Several syntaxes are supported:

- specifying `all=`<COLOR-INTERPRETATION>: set the color interpretation on all bands.
- specifying as many <COLOR-INTERPRETATION> as there are bands, starting from band 1 to the last band.
- specifying the color interpretation on a subset of bands with <BAND>=<COLOR-INTERPRETATION> where band is the band number starting at 1.

--scale [<BAND>=]<SCALE>

Added in version 3.13.

Set band scale factor. May be repeated.

Several syntaxes are supported:

- specifying one single <SCALE>: set the scale on all bands.
- specifying as many <SCALE> as there are bands, starting from band 1 to the last band.
- specifying the scale on a subset of bands with <BAND>=<SCALE> where band is the band number starting at 1.

--offset [<BAND>=]<OFFSET>

Added in version 3.13.

Set band offset constant. May be repeated.

Several syntaxes are supported:

- specifying one single <OFFSET>: set the offset on all bands.
- specifying as many <OFFSET> as there are bands, starting from band 1 to the last band.
- specifying the offset on a subset of bands with <BAND>=<OFFSET> where band is the band number starting at 1.

--metadata <KEY>=<VALUE>

Add/update metadata item, at the dataset level. May be repeated.

--nodata <value>

Override nodata value.

`null` or `none` can be specified to unset an existing nodata value.

--stats

Compute raster band statistics for all bands.

--unset-color-table

Added in version 3.13.

Remove the color table associated to the first band, or the one specified by *--band*.

--unset-metadata <KEY>

Remove metadata item, at the dataset level. May be repeated.

--unset-metadata-domain <DOMAIN>

Added in version 3.12.

Remove metadata domain, at the dataset level. May be repeated.

Standard Options

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Override (without reprojecting) the CRS of a dataset

```
$ gdal raster edit --crs=EPSG:32632 my.tif
```

Example 2: Override (without reprojecting or subsetting) the bounding box of a dataset

```
$ gdal raster edit --bbox=2,49,3,50 my.tif
```

Example 3: Add a metadata item

```
$ gdal raster edit --metadata AUTHOR=EvenR my.tif
```

Example 4: Remove a metadata item

```
$ gdal raster edit --unset-metadata AUTHOR my.tif
```

Example 5: Add 2 ground control point (GCP) for (column=0,line=0,X=2,Y=49) and (column=50,line=100,X=3,Y=48)

```
$ gdal raster edit --gcp 0,0,2,49 --gcp 50,100,3,48 my.tif
```

Example 6: Add ground control point (GCP) from gcps.csv, that must have fields named column, line, x and y.

```
$ gdal raster edit --gcp @gcps.csv my.tif
```

Example 7: Set red, green, blue, NIR color interpretation to a 4-band dataset

```
$ gdal raster edit --color-interpretation red,green,blue,nir 4band.tif
```

5.1.3.15 gdal raster footprint

Added in version 3.11.

Synopsis

```
Usage: gdal raster footprint [OPTIONS] <INPUT>... <OUTPUT>

Compute the footprint of a raster dataset.

Positional arguments:
  -i, --input <INPUT>                                    Input raster datasets [may be␣
↪repeated] [required] [not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
```

(continues on next page)

(continued from previous page)

```
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format [not available in␣
↪pipelines]
  --co, --creation-option <KEY>=<VALUE>                Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>         Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --append                                             Whether appending to existing␣
↪layer is allowed [not available in pipelines]
  --overwrite                                          Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --output-layer <OUTPUT-LAYER>                        Output layer name (default:␣
↪footprint)
  -b, --band <BAND>                                    Input band(s) (1-based index)␣
↪[may be repeated]
  --combine-bands <COMBINE-BANDS>                      Defines how the mask bands of the␣
↪selected bands are combined to generate a single mask band, before being vectorized..␣
↪COMBINE-BANDS=union|intersection (default: union)
  --overview <OVERVIEW>                                Which overview level of source␣
↪file must be used
                                                       Mutually exclusive with --input-
↪nodata
  --input-nodata <INPUT-NODATA>                        Set nodata values for input bands.
↪ [1.. values]
                                                       Mutually exclusive with --overview
  --coordinate-system <COORDINATE-SYSTEM>              Target coordinate system.␣
↪COORDINATE-SYSTEM=georeferenced|pixel
  --output-crs <OUTPUT-CRS>                            Output CRS
  --split-multipolygons                                Whether to split multipolygons as␣
↪several features each with one single polygon
  --convex-hull                                        Whether to compute the convex␣
↪hull of the footprint
  --densify-distance <DENSIFY-DISTANCE>                Maximum distance between 2␣
↪consecutive points of the output geometry.
  --simplify-tolerance <SIMPLIFY-TOLERANCE>            Tolerance used to merge␣
↪consecutive points of the output geometry.
  --min-ring-area <MIN-RING-AREA>                      Minimum value for the area of a␣
↪ring
  --max-points <MAX-POINTS>                            Maximum number of points of each␣
↪output geometry (default: 100)
  --location-field <LOCATION-FIELD>                    Name of the field where the path␣
↪of the input dataset will be stored. (default: location)
                                                       Mutually exclusive with --no-
↪location-field
  --no-location-field                                  Disable creating a field with the␣
↪path of the input dataset
                                                       Mutually exclusive with --
↪location-field
  --absolute-path                                      Whether the path to the input␣
↪dataset should be stored as an absolute path
```

(continues on next page)

(continued from previous page)

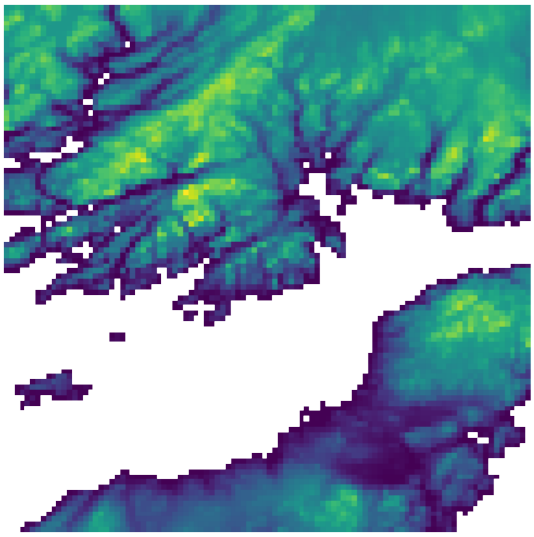
```
Advanced Options:
  --oo, --open-option <KEY>=<VALUE>                    Open options [may be repeated]␣
↪[not available in pipelines]
  --if, --input-format <INPUT-FORMAT>                  Input formats [may be repeated]␣
↪[not available in pipelines]
```

Description

gdal raster footprint computes the footprint of a raster file, taking into account nodata values (or more generally the mask band attached to the raster bands), and generating polygons/multipolygons corresponding to areas where pixels are valid, and write to an output vector file.

Since GDAL 3.12, this algorithm can be part of a *gdal pipeline*.

The **gdal raster clean-collar** utility may be run as a pre-processing step to generate proper mask bands.



Example footprint calculated with default options of **gdal raster footprint**.

Post-vectorization geometric operations are applied in the following order:

- optional splitting (*--split-multipolygons*)
- optional densification (*--densify-distance*)
- optional reprojection (*--output-crs*)
- optional filtering by minimum ring area (*--min-ring-area*)
- optional application of convex hull (*--convex-hull*)
- optional simplification (*--simplify-tolerance*)
- limitation of number of points (*--max-points*)

Added in version 3.12.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

> **Note**
>
> However this algorithm is not natively streaming compatible. Consequently a in-memory temporary dataset will be generated, which may cause significant processing time at opening.

Program-Specific Options

--absolute-path

Enables writing the absolute path of the input dataset. By default, the filename is written in the location field exactly as specified on the command line.

-b, --band <band>

Select an input <band> to be processed. Bands are numbered from 1. Multiple *-b* switches may be used to select a set of input bands. If not specified, all bands are taken into account. The way multiple bands are combined is controlled by *--combine-bands*

--combine-bands `union|intersection`

Defines how the mask bands of the selected bands are combined to generate a single mask band, before being vectorized. The default value is `union`: that is a pixel is valid if it is valid at least for one of the selected bands. `intersection` means that a pixel is valid only if it is valid for all selected bands.

--coordinate-system `pixel|georeferenced`

Target coordinate system. By default if the input dataset is georeferenced, `georeferenced` is implied, that is the footprint geometry will be expressed as coordinates in the CRS of the raster (or the one specified with *--output-crs*). If specifying `pixel`, the coordinates of the footprint geometry are column and line indices.

--convex-hull

When specified, the convex hull of (multi)polygons is computed.

--densify-distance <value>

The specified value of this option is the maximum distance between 2 consecutive points of the output geometry. The unit of the distance is in pixels if *--coordinate-system* equals `pixel`, or otherwise in georeferenced units of the source raster. This option is applied before the reprojection implied by *--output-crs*.

--input-nodata <value>

Set nodata values for input bands (different values can be supplied for each band). If a single value is specified, it applies to all selected bands. If more than one value is supplied, there should be as many values as the number of selected bands, and all values should be quoted to keep them together as a single operating system argument. If the option is not specified, the intrinsic mask band of each selected bands will be used.

--location-field <field_name>

Specifies the name of the field in the resulting vector dataset where the path of the input dataset will be stored. The default field name is "location". To prevent writing the path of the input dataset, use *--no-location-field*

--max-points <value>|`unlimited`

Maximum number of points of each output geometry (not counting the closing point of each ring, which is always identical to the first point). The default value is 100. `unlimited` can be used to remove that limitation.

--min-ring-area <value>

Minimum value for the area of a ring The unit of the area is in square pixels if *--coordinate-system* equals `pixel`, or otherwise in georeferenced units of the target vector dataset. This option is applied after the reprojection implied by *--output-crs*

--no-location-field

Turns off the writing of the path of the input dataset as a field in the output vector dataset.

--output-crs <CRS_DEF>

Target CRS of the output file. The <CRS_DEF> may be any of the usual GDAL/OGR forms, complete WKT, PROJ.4, EPSG:n or a file containing the WKT. Specifying this option implies `--coordinate-system=georeferenced`. The footprint is reprojected from the CRS of the source raster to the specified CRS.

--output-layer <OUTPUT-LAYER>

Output layer name. Defaults to `footprint`.

--overview <index>

To specify which overview level of source file must be used, when overviews are available on the source raster. By default the full resolution level is used. The index is 0-based, that is 0 means the first overview level. This option is mutually exclusive with *--input-nodata*.

--simplify-tolerance <value>

The specified value of this option is the tolerance used to merge consecutive points of the output geometry using the *OGRGeometry::Simplify()* method. The unit of the distance is in pixels if *--coordinate-system* equals `pixel`, or otherwise in georeferenced units of the target vector dataset. This option is applied after the reprojection implied by *--output-crs*.

--split-multipolygons

When specified, multipolygons are split as several features each with one single polygon.

Standard Options

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Write the footprint of a GeoTIFF file into a GeoJSON file.

```
gdal raster footprint my_raster.tif footprint.geojson
```

5.1.3.16 gdal raster fill-nodata

Added in version 3.11.

Synopsis

```
Usage: gdal raster fill-nodata [OPTIONS] <INPUT> <OUTPUT>

Fill nodata raster regions by interpolation from edges.

Positional arguments:
  -i, --input <INPUT>                                    Input raster datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output raster dataset [required]␣
↪[not available in pipelines]

Common Options:
```

(continues on next page)

(continued from previous page)

```
  -h, --help                                          Display help message and exit
  --json-usage                                        Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                              Configuration option [may be␣
↪repeated]
  -q, --quiet                                         Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>               Creation option [may be repeated]␣
↪[not available in pipelines]
  --overwrite                                         Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                      Mutually exclusive with --append
  --append                                            Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                      Mutually exclusive with --
↪overwrite
  -b, --band <BAND>                                   Input band (1-based index)␣
↪(default: 1)
  -d, --max-distance <MAX_DISTANCE>                   The maximum distance (in pixels)␣
↪that the algorithm will search out for values to interpolate. (default: 100)
  -s, --smoothing-iterations <SMOOTHING_ITERATIONS>    The number of 3x3 average filter␣
↪smoothing iterations to run after the interpolation to dampen artifacts. The default␣
↪is zero smoothing iterations. (default: 0)
  --mask <MASK>                                       Use the first band of the␣
↪specified file as a validity mask (zero is invalid, non-zero is valid).
  --strategy <STRATEGY>                               By default, pixels are␣
↪interpolated using an inverse distance weighting (invdist). It is also possible to␣
↪choose a nearest neighbour (nearest) strategy.. STRATEGY=invdist|nearest (default:␣
↪invdist)

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                 Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                   Open options [may be repeated]␣
↪[not available in pipelines]
```

Description

gdal raster fill-nodata fills nodata areas by interpolating from valid pixels around the edges of the area.

This subcommand is also available as a potential step of *gdal raster pipeline* (since GDAL 3.12)

Added in version 3.12.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

> **Note**
>
> However this algorithm is not natively streaming compatible. Consequently a temporary dataset will be generated, which may cause significant processing time at opening.

Program-Specific Options

-b, --band <BAND>

Select an input <BAND> to be processed. Bands are numbered from 1. Default is the first band of the input dataset.

--mask <MASK>

Use the first band of the specified file as a validity mask (zero is invalid, non-zero is valid).

--max-distance <MAX_DISTANCE>

Specifies the maximum distance (in pixels) that the algorithm will search out for values to interpolate. Default is 100 pixels.

--smoothing-iterations <SMOOTHING_ITERATIONS>

Specifies the number of smoothing iterations to apply to the filled raster. This can help to reduce artifacts in the filled areas. Default is 0 iterations.

--strategy <STRATEGY>

Select the interpolation <STRATEGY> to use. By default, pixels are interpolated using an inverse distance weighting (*invdist*). It is also possible to choose a nearest neighbour (*nearest*) strategy.

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Fill nodata areas in a raster

The command specifies to use the second band of the input raster, 50 px max distance, 3 smoothing iterations and the *nearest* strategy for interpolation. The output will be saved in *output.tif*.

```
gdal raster fill-nodata -b 2 --max-distance 50 --smoothing-iterations 3 \
    --strategy nearest --mask mask.tif \
    input.tif output.tif
```

5.1.3.17 gdal raster hillshade

Added in version 3.11.

Synopsis

```
Usage: gdal raster hillshade [OPTIONS] <INPUT> <OUTPUT>

Generate a shaded relief map

Positional arguments:
  -i, --input <INPUT>                                    Input raster datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output raster dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
```

(continues on next page)

(continued from previous page)

```
↪[not available in pipelines]
  --overwrite                                           Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                        Mutually exclusive with --append
  --append                                              Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                        Mutually exclusive with --
↪overwrite
  -b, --band <BAND>                                     Input band (1-based index)␣
↪(default: 1)
  -z, --zfactor <ZFACTOR>                               Vertical exaggeration used to pre-
↪multiply the elevations
  --xscale <XSCALE>                                     Ratio of vertical units to␣
↪horizontal X axis units
  --yscale <YSCALE>                                     Ratio of vertical units to␣
↪horizontal Y axis units
  --azimuth <AZIMUTH>                                   Azimuth of the light, in degrees␣
↪(default: 315)
  --altitude <ALTITUDE>                                 Altitude of the light, in degrees␣
↪(default: 45)
  --gradient-alg <GRADIENT-ALG>                         Algorithm used to compute terrain␣
↪gradient. GRADIENT-ALG=Horn|ZevenbergenThorne (default: Horn)
  --variant <VARIANT>                                   Variant of the hillshading␣
↪algorithm. VARIANT=regular|combined|multidirectional|Igor (default: regular)
  --no-edges                                            Do not try to interpolate values␣
↪at dataset edges or close to nodata values

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                   Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                     Open options [may be repeated]␣
↪[not available in pipelines]
```

Description

gdal raster hillshade generates a shaded relief map, from any GDAL-supported elevation raster.

This subcommand is also available as a potential step of *gdal raster pipeline*

It generates an 8-bit raster with a nice shaded relief effect. It is very useful for visualizing the terrain. You can optionally specify the azimuth and altitude of the light source, a vertical exaggeration factor and scaling factors to account for differences between vertical and horizontal units.

The value 0 is used as the output nodata value. A nodata value in the target dataset will be emitted if at least one pixel set to the nodata value is found in the 3x3 window centered around each source pixel. By default, the algorithm will compute values at image edges or if a nodata value is found in the 3x3 window, by interpolating missing values, unless `--no-edges` is specified, in which case a 1-pixel border around the image will be set with the nodata value.

In general, it assumes that x, y and z units are identical. However, if none of `--xscale` and `--yscale` are specified, and the CRS is a geographic or projected CRS, it will automatically determine the appropriate ratio from the units of the CRS, as well as the potential value of the units of the raster band (as returned by `GDALRasterBand::GetUnitType()`, if it is metre, foot international or US survey foot). Note that for geographic CRS, the result for source datasets at high latitudes may be incorrect, and prior reprojection to a polar projection might be needed using *gdal raster reproject*.

If x (east-west) and y (north-south) units are identical, but z (elevation) units are different, the `--xscale` and `--yscale`

can be used to set the ratio of vertical units to horizontal. For geographic CRS near the equator, where units of latitude and units of longitude are similar, elevation (z) units can be converted to be compatible by using scale=370400 (if elevation is in feet) or scale=111120 (if elevation is in meters). For locations not near the equator, the `--xscale` value can be taken as the `--yscale` value multiplied by the cosine of the mean latitude of the raster.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

--altitude <ALTITUDE>

Altitude of the light, in degrees. 90 if the light comes from above the DEM, 0 if it is raking light. The default value is 45 degree.

This option is mutually exclusive with `--variant=Igor`.

--azimuth <AZIMUTH>

Azimuth of the light, in degrees. 0 if it comes from the top of the raster, 90 from the east, ... The default value, 315, should rarely be changed as it is the value generally used to generate shaded maps.

This option is mutually exclusive with `--variant=multidirectional`.

-b, --band <BAND>

Index (starting at 1) of the band to which the hillshade must be computed.

--gradient-alg `Horn|ZevenbergenThorne`

Algorithm used to compute terrain gradient. The default is `Horn`. The literature suggests Zevenbergen & Thorne to be more suited to smooth landscapes, whereas Horn's formula to perform better on rougher terrain.

--no-edges

Do not try to interpolate values at dataset edges or close to nodata values

--variant `regular|combined|multidirectional|Igor`

Variant of the hillshading algorithm:

- `regular`: the hillshade values combines the computed slope with the azimuth and altitude of the illumination according to:

$$Hillshade = 1 + 254.0 * ((\sin(altitude) * cos(slope)) + (cos(altitude) * sin(slope) * cos(azimuth - \frac{\pi}{2} - aspec$$

- `combined`: combined shading, a combination of slope and oblique shading.
- `multidirectional`: multidirectional shading, a combination of hillshading illuminated from 225 deg, 270 deg, 315 deg, and 360 deg azimuth. Applies the formula of http://pubs.usgs.gov/of/1992/of92-422/of92-422.pdf
- `Igor`: shading which tries to minimize effects on other map features beneath. Igor's hillshading uses formula from Maperitive: http://maperitive.net/docs/Commands/GenerateReliefImageIgor.html

--xscale <scale>

Added in version 3.11.

Ratio of vertical units to horizontal X axis units. If the horizontal unit of the source DEM is degrees (e.g Lat/Long WGS84 projection), you can use scale=111120 if the vertical units are meters (or scale=370400 if they are in feet).

If none of *--xscale* and *--yscale* are specified, and the CRS is a geographic or projected CRS, **gdal raster hillshade** will automatically determine the appropriate ratio from the units of the CRS, as well as the potential value of the units of the raster band (as returned by *GDALRasterBand::GetUnitType()*, if it is metre, foot international or US survey foot). Note that for geographic CRS, the result for source datasets at high latitudes may be incorrect, and prior reprojection to a polar projection might be needed.

If *--xscale* is specified, *--yscale* must also be specified.

--yscale <scale>

Added in version 3.11.

Ratio of vertical units to horizontal Y axis units. If the horizontal unit of the source DEM is degrees (e.g Lat/Long WGS84 projection), you can use scale=111120 if the vertical units are meters (or scale=370400 if they are in feet)

If none of *--xscale* and *--yscale* are specified, and the CRS is a geographic or projected CRS, **gdal raster hillshade** will automatically determine the appropriate ratio from the units of the CRS, as well as the potential value of the units of the raster band (as returned by *GDALRasterBand::GetUnitType()*, if it is metre, foot international or US survey foot). Note that for geographic CRS, the result for source datasets at high latitudes may be incorrect, and prior reprojection to a polar projection might be needed.

If *--yscale* is specified, *--xscale* must also be specified.

-z, --zfactor <ZFACTOR>

Vertical exaggeration used to pre-multiply the elevations

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Generates a shaded relief map from a DTED0 file, using a vertical exaggeration factor of 30.

```
$ gdal raster hillshade --zfactor=30 n43.dt0 out.tif --overwrite
```

Example 2: Combine the output of shaded relief map and hypsometric rendering on a DEM to create a colorized shaded relief map.

```
$ gdal pipeline read n43.tif ! \
                color-map --color-map color_file.txt ! \
                blend --operator=hsv-value --overlay \
                    [ read n43.tif ! hillshade -z 30 ] ! \
                write out.tif --overwrite
```

5.1.3.18 gdal raster index

Added in version 3.11.

Synopsis

```
Usage: gdal raster index [OPTIONS] <INPUT>... <OUTPUT>

Create a vector index of raster datasets.

Positional arguments:
  -i, --input <INPUT>                                Input raster datasets [may be␣
↪repeated] [required]
  -o, --output <OUTPUT>                              Output vector dataset [required]

Common Options:
  -h, --help                                         Display help message and exit
  --json-usage                                       Display usage as JSON document␣
↪and exit
```

(continues on next page)

(continued from previous page)

```
  --config <KEY>=<VALUE>                                  Configuration option [may be␣
↪repeated]
  -q, --quiet                                             Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format
  --co, --creation-option <KEY>=<VALUE>                   Creation option [may be repeated]
  --lco, --layer-creation-option <KEY>=<VALUE>            Layer creation option [may be␣
↪repeated]
  --overwrite                                             Whether overwriting existing␣
↪output dataset is allowed
                                                          Mutually exclusive with --update
  --update                                                Whether to open existing dataset␣
↪in update mode
                                                          Mutually exclusive with --
↪overwrite
  --overwrite-layer                                       Whether overwriting existing␣
↪output layer is allowed
  --append                                                Whether appending to existing␣
↪layer is allowed
  --nln, --output-layer <OUTPUT-LAYER>                    Output layer name
  --recursive                                             Whether input directories should␣
↪be explored recursively.
  --filename-filter <FILENAME-FILTER>                     Pattern that the filenames in␣
↪input directories should follow ('*' and '?' wildcard) [may be repeated]
  --min-pixel-size <MIN-PIXEL-SIZE>                       Minimum pixel size in term of␣
↪geospatial extent per pixel (resolution) that a raster should have to be selected.
  --max-pixel-size <MAX-PIXEL-SIZE>                       Maximum pixel size in term of␣
↪geospatial extent per pixel (resolution) that a raster should have to be selected.
  --location-name <LOCATION-NAME>                         Name of the field with the raster␣
↪path (default: location)
  --absolute-path                                         Whether the path to the input␣
↪datasets should be stored as an absolute path
  --output-crs <OUTPUT-CRS>                               Output CRS
  --metadata <KEY>=<VALUE>                                Add dataset metadata item [may be␣
↪repeated]
  --skip-errors                                           Skip errors related to input␣
↪datasets
  --profile <PROFILE>                                     Profile of output dataset.␣
↪PROFILE=none|STAC-GeoParquet (default: none)
  --base-url <BASE-URL>                                   Base URL for STAC-GeoParquet href
  --id-method <ID-METHOD>                                 How to derive STAC-GeoParquet 'id
↪'. ID-METHOD=filename|md5|metadata-item (default: filename)
  --id-metadata-item <ID-METADATA-ITEM>                   Name of metadata item used to set␣
↪STAC-GeoParquet 'id' (default: id)
  --source-crs-field-name <SOURCE-CRS-FIELD-NAME>         Name of the field to store the␣
↪CRS of each dataset
  --source-crs-format <SOURCE-CRS-FORMAT>                 Format in which the CRS of each␣
↪dataset must be written. SOURCE-CRS-FORMAT=auto|WKT|EPSG|PROJ (default: auto)
```

Description

gdal raster index creates a vector dataset with a record for each input raster file, an attribute containing the filename, and a polygon geometry outlining the raster. This output is suitable for use with MapServer as a raster tileindex

See *gdal driver gti create* for an extension of this command that create files to be used as input for the *GTI* driver.

The following options are available:

Program-Specific Options

--absolute-path

The absolute path to the raster files is stored in the index file. By default the raster filenames will be put in the file exactly as they are specified on the command line.

--base-url <BASE-URL>

Added in version 3.13.

Base URL for STAC-GeoParquet the `href` property of the *id* property in a STAC-GeoParquet index.

--filename-filter <FILENAME-FILTER>

Pattern that the filenames contained in input directories should follow. '*' is a wildcard character that matches any number of any characters including none. '?' is a wildcard character that matches a single character. Comparisons are done in a case insensitive way. Several filters may be specified.

For example: `--filename-filter "*.tif,*.tiff"`

--id-metadata-item <ID-METADATA-ITEM>

Added in version 3.13.

Name of metadata item used to set STAC-GeoParquet *id* property. Defaults to `id`.

--id-method `filename|md5|metadata-item`

Added in version 3.13.

How to derive STAC-GeoParquet *id* property

- `filename` (the default): only the filename (without directory, but including extension) is used
- `md5`: the MD5Sum of the file is computed, and suffixed with the filename.
- `metadata-item`: the value of the metadata item defined by *--id-metadata-item* from each source dataset is used.

--location-name <LOCATION-NAME>

The output field name to hold the file path/location to the indexed rasters. The default field name is `location`.

--max-pixel-size <val>

Maximum pixel size in term of geospatial extent per pixel (resolution) that a raster should have to be selected. The pixel size is evaluated after reprojection of its extent to the target CRS defined by *--output-crs*.

--metadata <KEY>=<VALUE>

Write an arbitrary layer metadata item, for formats that support layer metadata. This option may be repeated.

--min-pixel-size <val>

Minimum pixel size in term of geospatial extent per pixel (resolution) that a raster should have to be selected. The pixel size is evaluated after reprojection of its extent to the target CRS defined by *--output-crs*.

--output-crs <OUTPUT-CRS>

Geometries of input files will be transformed to the desired target coordinate reference system. Default creates simple rectangular polygons in the same coordinate reference system as the input rasters.

--profile none|STAC-GeoParquet

Added in version 3.13.

Defines the profile of the output dataset. If selecting STAC-GeoParquet, a Parquet file conforming to the STAC-GeoParquet specification is created (provided that the Parquet format is also selected). Such file can be read by the *GTI driver*. Setting STAC-GeoParquet also implicitly sets the target CRS to EPSG:4326.

--recursive

Whether input directories should be explored recursively.

--skip-errors

Added in version 3.12.

Skip errors related to input datasets.

--source-crs-field-name <SOURCE-CRS-FIELD-NAME>

The name of the field to store the CRS of each tile. This field name can be used as the value of the TILESRS keyword in MapServer

--source-crs-format auto|WKT|EPSG|PROJ

The format in which the CRS of each tile must be written. Types can be auto, WKT, EPSG, PROJ. This option should be used together with *--source-crs-field-name*.

Standard Options

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--nln, --output-layer <LAYER>

Provides a name for the output vector layer.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--update

Whether the output dataset must be opened in update mode. Implies that it already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1

Produce a GeoPackage with a record for every image that the utility found in the `doq` folder. Each record holds information that points to the location of the image and also a bounding rectangle shape showing the bounds of the image:

```
gdal raster index doq/*.tif doq_index.gpkg
```

Example 2

The *--output-crs* option can also be used to transform all input raster geometries into the same output projection:

```
gdal raster index --output-crs EPSG:4326 --source-crs-field-name=src_srs *.tif tile_
↪index_mixed_crs.gpkg
```

Example 3: Creates a STAC-GeoParquet compliant index.

```
gdal raster index --profile STAC-GeoParquet  *.tif index.parquet
```

5.1.3.19 gdal raster materialize

Added in version 3.12.

Description

The `materialize` operation is for use only in a pipeline, and writes the incoming dataset to a dataset on disk, before passing that materialized dataset to following steps. This can be used when materializing a dataset helps for performance compared to on-demand evaluation of previous steps.

By default, a tiled compressed GeoTIFF file is used for the materialized dataset, created in the directory pointed by the *CPL_TMPDIR* configuration option or in the current directory if not specified, and is deleted when the pipeline finishes. The user can also select another format, or specify an explicit filename, in which case the materialized dataset is not deleted when the pipeline finishes.

> **Note**
>
> VRT output is not compatible with materialize

Synopsis

```
* materialize [OPTIONS]
-----------------------

Materialize a piped dataset on disk to increase the efficiency of the following steps.

Options:
  -o, --output <OUTPUT>                                Materialized dataset name␣
↪(created by algorithm)
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format
  --co, --creation-option <KEY>=<VALUE>                Creation option [may be repeated]
  --overwrite                                          Whether overwriting existing␣
↪output dataset is allowed
```

Program-Specific Options

--output <OUTPUT>

Optional dataset name. When specified, it is not removed at the end of the process.

Standard Options

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Reproject a GeoTIFF file to CRS EPSG:32632 ("WGS 84 / UTM zone 32N"), materialize it to a temporary file and compute its contour lines

```
$ gdal pipeline ! read in.tif ! reproject --output-crs=EPSG:32632 ! \
                ! materialize ! contour --interval=10 ! write out.gpkg --overwrite
```

Example 2: How to use materialize for Cloud Optimized GeoTIFF (COG)

Usually when you want load COG data, you are not loading the whole data but instead on certain region and using lower resolution for fast analysis. Therefore, after using command *read*, it is not supposed to be materialize right away because it mean it will download the whole thing into the memory/disk. Instead, you could use command *reproject* and use *--bbox* to limit the region of interest and set *--size* or *--resolution* to use lower resolution overview.

```
$ gdal pipeline ! read /vsicurl/https://some.storage.com/mydata/landcover.tif \
                ! reproject -r mode -d EPSG:4326 --bbox=112,2,116,4.5 --bbox-
↪crs=EPSG:4326 --size=3000,3000 \
                ! materialize \
                ! zonal-stats --zones=/vsicurl/https://some.storage.com/mydata/adm_
↪level4.fgb --stat=values \
                ! write out.geojson
```

5.1.3.20 gdal raster mosaic

Added in version 3.11.

Synopsis

```
Usage: gdal raster mosaic [OPTIONS] <INPUTS>... <OUTPUT>

Build a mosaic, either virtual (VRT) or materialized.

Positional arguments:
  -i, --input <INPUTS>                                          Input raster datasets␣
↪(or specify a @<filename> to point to a file containing filenames) [1.. values]␣
↪[required] [not available in pipelines]
  -o, --output <OUTPUT>                                         Output raster dataset␣
↪[required] [not available in pipelines]

Common Options:
  -h, --help                                                    Display help message␣
↪and exit
  --json-usage                                                  Display usage as JSON␣
↪document and exit
  --config <KEY>=<VALUE>                                        Configuration option␣
↪[may be repeated]
  -q, --quiet                                                   Quiet mode (no␣
↪progress bar or warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>           Output format ("GDALG"␣
```

(continues on next page)

(continued from previous page)

```
↪allowed) [not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                          Creation option [may␣
↪be repeated] [not available in pipelines]
  --overwrite                                                    Whether overwriting␣
↪existing output dataset is allowed [not available in pipelines]
                                                                 Mutually exclusive␣
↪with --append
  --append                                                       Append as a subdataset␣
↪to existing output [not available in pipelines]
                                                                 Mutually exclusive␣
↪with --overwrite
  -b, --band <BAND>                                              Input band(s) (1-based␣
↪index) [may be repeated]
  --absolute-path                                                Whether the path to␣
↪the input datasets should be stored as an absolute path
  --resolution <xres>,<yres>|same|average|common|highest|lowest>  Target resolution (in␣
↪destination CRS units) (default: same)
  --bbox <BBOX>                                                  Target bounding box as␣
↪xmin,ymin,xmax,ymax (in destination CRS units)
  --target-aligned-pixels                                        Round target extent to␣
↪target resolution
  --input-nodata <INPUT-NODATA>                                  Set nodata values for␣
↪input bands. [1.. values]
  --output-nodata <OUTPUT-NODATA>                                Set nodata values at␣
↪the destination band level. [1.. values]
  --hide-nodata                                                  Makes the destination␣
↪band not report the NoData.
  --add-alpha                                                    Adds an alpha mask␣
↪band to the destination when the source raster have none.
  --pixel-function <PIXEL-FUNCTION>                              Specify a pixel␣
↪function to calculate output value from overlapping inputs. PIXEL-
↪FUNCTION=abs|area|argmax|argmin|cmul|complex|conj|dB|dB2amp|dB2pow|diff|div|exp|expression|geometric_
↪mean|harmonic_mean|imag|intensity|interpolate_exp|interpolate_
↪linear|inv|log10|max|mean|median|min|mod|mode|mul|norm_
↪diff|phase|polar|pow|quantile|real|reclassify|replace_nodata|round|scale|sqrt|sum
  --pixel-function-arg <NAME>=<VALUE>                            Specify argument(s) to␣
↪pass to the pixel function [may be repeated]

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                            Input formats [may be␣
↪repeated] [not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                              Open options [may be␣
↪repeated] [not available in pipelines]
```

Description

This program builds a mosaic of a list of input GDAL datasets, that can be either a virtual mosaic in the *VRT (Virtual Dataset)* format, or in a more conventional raster format such as GeoTIFF.

Wildcards '*', '?' or '[']' of *VSIGlob()* can be used for input dataset names, even on files located on network file systems such as /vsis3/, /vsigs/, /vsiaz/, etc. Alternatively if a input dataset name is prefixed by the @ character, it will be assumed to contain the list of actual dataset names (one per line) to use as input datasets.

gdal raster mosaic does some checks to ensure that all files that will be put in the resulting file have similar characteristics: number of bands, projection, color interpretation, etc. If not, files that do not match the common characteristics will be skipped.

Starting with GDAL 3.12, a function (e.g., `min`, `mean`, `median`) can be specified (*--pixel-function*) to calculate pixel values from overlapping inputs. If no function is specified, or in earlier versions, the order of the input list is used to determine priority. Files that are listed at the end are the ones from which the content will be fetched. Note that nodata will be taken into account to potentially fetch data from lower-priority datasets, but currently, alpha channel is not taken into account to do alpha compositing (so a source with alpha=0 appearing on top of another source will override its content). This might be changed in later versions.

Stating with GDAL 3.12, this command can also be used as the first step of *gdal raster pipeline*.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

--absolute-path

Added in version 3.12.0.

When writing a VRT file, enables writing the absolute path of the input datasets. By default, input filenames are written in a relative way with respect to the VRT filename (when possible).

--add-alpha

Adds an alpha mask band to the output when the source raster have none. Mainly useful for RGB sources (or grey-level sources). The alpha band is filled on-the-fly with the value 0 in areas without any source raster, and with value 255 in areas with source raster. The effect is that a RGBA viewer will render the areas without source rasters as transparent and areas with source rasters as opaque.

-b, --band <band>

Select an input <band> to be processed. Bands are numbered from 1. If input bands not set all bands will be added to the output. Multiple *-b* switches may be used to select a set of input bands.

--bbox <xmin>,<ymin>,<xmax>,<ymax>

Set georeferenced extents of output file. The values must be expressed in georeferenced units. If not specified, the extent of the output is the minimum bounding box of the set of source rasters. Pixels within the extent of the output but not covered by a source raster will be read as valid pixels with a value of zero unless a NODATA value is specified using *--output-nodata* or an alpha mask band is added with *--add-alpha*.

--hide-nodata

Even if any band contains nodata value, giving this option makes the output band not report the NoData. Useful when you want to control the background color of the dataset. By using along with the *--add-alpha* option, you can prepare a dataset which doesn't report nodata value but is transparent in areas with no data.

--input-nodata <value>[,<value>]...

Set nodata values for input bands (different values can be supplied for each band). If the option is not specified, the intrinsic nodata settings on the source datasets will be used (if they exist). The value set by this option is written in the NODATA element of each `ComplexSource` element.

--output-nodata <value>[,<value>]...

Set nodata values at the output band level (different values can be supplied for each band). If more than one value is supplied, all values should be quoted to keep them together as a single operating system argument (*Example 1*). If the option is not specified, intrinsic nodata settings on the first dataset will be used (if they exist). The

value set by this option is written in the `NoDataValue` element of each `VRTRasterBand element`. Use a value of *None* to ignore intrinsic nodata settings on the source datasets.

--pixel-function

Specify a function name to calculate a value from overlapping inputs. For a list of available pixel functions, see *Built-in Pixel Functions*. If no function is specified, values will be taken from the last overlapping input.

Added in version 3.12.

--pixel-function-arg

Specify an argument to be provided to a pixel function, in the format `<NAME>=<VALUE>`. Multiple arguments may be specified by repeating this option.

Added in version 3.12.

--resolution `{<xres,yres>|same|highest|lowest|average}`

In case the resolution of all input files is not the same, the `--resolution` flag enables the user to control the way the output resolution is computed.

same, the default, checks that all source rasters have the same resolution and errors out when this is not the case.

highest will pick the smallest values of pixel dimensions within the set of source rasters.

lowest will pick the largest values of pixel dimensions within the set of source rasters.

average will compute an average of pixel dimensions within the set of source rasters.

common determines the greatest common divisor of the source pixel dimensions, e.g. 0.2 for source pixel dimensions of 0.4 and 0.6.

<xres>,<yres>. The values must be expressed in georeferenced units. Both must be positive values.

--target-aligned-pixels

(target aligned pixels) align the coordinates of the extent of the output file to the values of the `--resolution`, such that the aligned extent includes the minimum extent. Alignment means that xmin / resx, ymin / resy, xmax / resx and ymax / resy are integer values.

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option `<NAME>=<VALUE>`

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format `<format>`

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be

repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Make a virtual mosaic with blue background colour (RGB: 0 0 255)

```
gdal raster mosaic --hide-nodata --output-nodata=0,0,255 doq/*.tif doq_index.vrt
```

Example 2: Create a Cloud Optimized GeoTIFF (COG) mosaic from all GeoTIFFs in a folder

Because the size of the resulting GeoTIFF will be more than 4 GB, the *BIGTIFF=YES* creation option is used.

```
gdal raster mosaic --output-format COG --creation-option BIGTIFF=YES *.tif mosaic.tif
```

5.1.3.21 gdal raster neighbors

Added in version 3.12.

Synopsis

```
Usage: gdal raster neighbors [OPTIONS] <INPUT> <OUTPUT>

Compute the value of each pixel from its neighbors (focal statistics)

Positional arguments:
  -i, --input <INPUT>                                       Input raster datasets␣
↪[required] [not available in pipelines]
  -o, --output <OUTPUT>                                     Output raster dataset␣
↪[required] [not available in pipelines]

Common Options:
  -h, --help                                                Display help message and exit
  --json-usage                                              Display usage as JSON␣
```

(continues on next page)

(continued from previous page)

```
↪document and exit
  --config <KEY>=<VALUE>                                    Configuration option [may be␣
↪repeated]
  -q, --quiet                                               Quiet mode (no progress bar␣
↪or warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>       Output format ("GDALG"␣
↪allowed) [not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                     Creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                               Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                            Mutually exclusive with --
↪append
  --append                                                  Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                            Mutually exclusive with --
↪overwrite
  -b, --band <BAND>                                         Input band (1-based index)
  --method <METHOD>                                         Method to combine weighed␣
↪source pixels. METHOD=mean|sum|min|max|stddev|median|mode [may be repeated]
  --size <SIZE>                                             Neighborhood size
  --kernel <KERNEL>                                         Convolution kernel(s) to␣
↪apply [1.. values] [required]
  --ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>  Output data type. OUTPUT-DATA-
↪TYPE=UInt8|Int8|UInt16|Int16|UInt32|Int32|UInt64|Int64|CInt16|CInt32|Float16|Float32|Float64|CFloat32
↪(default: Float64)
  --nodata <NODATA>                                         Assign a specified nodata␣
↪value to output bands ('none', numeric value, 'nan', 'inf', '-inf')

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                       Input formats [may be␣
↪repeated] [not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                         Open options [may be␣
↪repeated] [not available in pipelines]
```

Description

gdal raster neighbors applies a kernel (convolution matrix) and a method to compute the target pixel value from a neighbourhood of the source pixel value.

At the top edge, the values of the first row are replicated to virtually extend the source window by a number of rows equal to the radius of the kernel. And similarly for the bottom, left and right edges. This strategy may potentially lead to unexpected results depending on the applied kernel.

For a given target cell, if the corresponding source value (the one at the center of the kernel) is nodata, the resulting pixel is nodata. For other source values in the neighborhood defined by the kernel, source nodata values are ignored.

This algorithm can be part of a *gdal pipeline* or *gdal raster pipeline*.

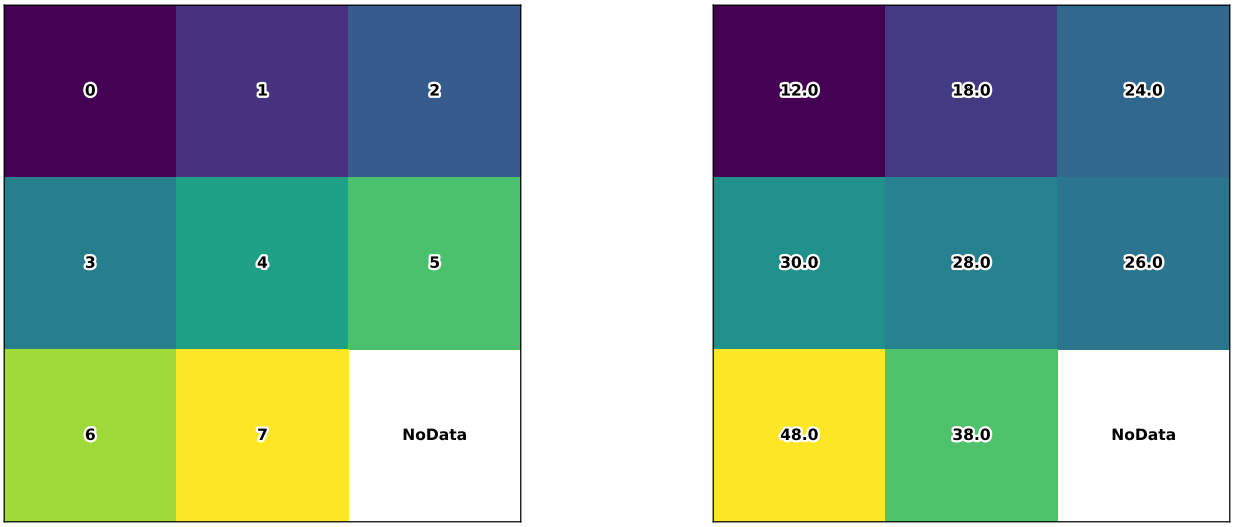


Fig. 1: Raster dataset before (left) and after (right) summation with a 3x3 equal-weight kernel. NoData values are considered zero for the purpose of the summation. Edge cells are replicated where the kernel window extends beyond the edge of the dataset.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

`--band` `<BAND>`

Source band number (indexing starts at one). If it is not specified, all input bands are used.

`--kernel`

Convolution kernel to apply to source pixels in a neighborhood of each pixel.

- either the name of a well-known kernel, among `edge1`, `edge2`, `sharpen`, `gaussian`, or `unsharp-masking`, corresponding to those kernels (convolution matrix), with the addition of:
  - `equal` corresponding to a kernel with all coefficients at one kernel size can be any odd number between 3 and 99):

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{5.1}$$

    For method `mean`, this corresponds to a box blur filter.
  - `u` corresponding to an horizontal derivative with coefficients:

$$\begin{bmatrix} 0 & 0 & 0 \\ -0.5 & 0 & 0.5 \\ 0 & 0 & 0 \end{bmatrix} \tag{5.2}$$

- v corresponding to a vertical derivative with coefficients:

$$\begin{bmatrix} 0 & -0.5 & 0 \\ 0 & 0 & 0 \\ 0 & 0.5 & 0 \end{bmatrix} \tag{5.3}$$

edge1, edge2, sharpen, u and v are only supported for a kernel size equal to 3.

gaussian is supported for a kernel size equal to 3 or 5.

unsharp-masking is supported for a kernel size equal to 5.

- or the values of the coefficients of the kernel as a square matrix of width and height N, where N is an odd number, as `[[val00, val01, ..., val0N],[val10, val11, ..., val1N],...,[valN0, valN1, ..., valNN]]`.

  If *--method* is set to mean, this has the effect of adding the sum of the (contributing, i.e. non nodata) weighted source pixels and dividing it by the sum of the coefficients in the kernel.

If *--kernel* is specified several times, there will one output band for each combination of kernel and input band.

--method sum|mean|min|max|stddev|median|mode

Function to apply to the weighted source pixels in the neighborhood defined by the kernel. Defaults to mean, except when *--kernel* is set to u, v, edge1, edge2, or a user defined kernel whose sum of coefficients is zero, in which case sum is used.

- sum: computes the sum of the value of contributing source pixels multiplied by the corresponding weight of the kernel. This corresponds to a kernel with un-normalized sum of coefficients.
- mean: computes the average of the value of contributing source pixels multiplied by the corresponding weight of the kernel. This has the effect of normalizing kernel coefficients so their sum is one.
- min: computes the minimum of the value of contributing source pixels multiplied by the corresponding weight of the kernel
- max: computes the maximum of the value of contributing source pixels multiplied by the corresponding weight of the kernel
- stddev: computes the standard deviation of the value of contributing source pixels multiplied by the corresponding weight of the kernel
- median: computes the median of the value of contributing source pixels multiplied by the corresponding weight of the kernel
- mode (majority): computes the most frequent of the value of contributing source pixels multiplied by the corresponding weight of the kernel

--nodata

Set the NoData value for the output dataset. May be set to none to leave the NoData value undefined. If *--nodata* is not specified, **gdal raster neighbors** will use a NoData value from the first source dataset to have one.

--size <SIZE>

Size of the kernel. Odd number between 3 and 99.

> **Note**
>
> Computation time is proportional to the square of the kernel size.

For kernels edge1, edge2, sharpen, u, v, gaussian, equal, defaults to 3.

For kernel `unsharp-masking`, defaults to 5.

For kernels specified through their coefficient values, it is deduced from the shape of the matrix.

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>

Output data type among `Byte`, `Int8`, `UInt16`, `Int16`, `UInt32`, `Int32`, `UInt64`, `Int64`, `CInt16`, `CInt32`, `Float32`, `Float64`, `CFloat32`, `CFloat64`.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Compute the horizontal and vertical derivative of a single-band raster

```
gdal raster neighbors --kernel u --kernel v in.tif uv.tif
```

Example 2: Compute the average value around each pixel in a 3x3 neighborhood

```
gdal raster neighbors --kernel equal --method mean in.tif mean.tif
```

Example 3: Compute the maximum value around each pixel in a 5x5 neighborhood

```
gdal raster neighbors --kernel equal --size 5 --method max in.tif max.tif
```

Example 4: Compute a sharpen filter of a single-band raster, by manually specifying the kernel coefficients.

```
gdal raster neighbors "--kernel=[[0,-1,0],[-1,5,-1],[0,-1,0]]" in.tif sharpen.tif
```

5.1.3.22 gdal raster nodata-to-alpha

Added in version 3.12.

Synopsis

```
Usage: gdal raster nodata-to-alpha [OPTIONS] <INPUT> <OUTPUT>

Replace nodata value(s) with an alpha band.

Positional arguments:
  -i, --input <INPUT>                                    Input raster datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output raster dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                         Mutually exclusive with --append
```

(continues on next page)

(continued from previous page)

```
  --append                                                Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                          Mutually exclusive with --
↪overwrite
  --nodata <NODATA>                                       Override nodata value of input␣
↪band(s) (numeric value, 'nan', 'inf', '-inf') [may be repeated]

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                     Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                       Open options [may be repeated]␣
↪[not available in pipelines]
```

Description

gdal raster nodata-to-alpha adds an alpha band based on the nodata value metadata set on the bands of the input dataset.

If there is no explicit nodata value on the input dataset, it is possible to add it with the *--nodata* option.

This program is a no-op and can be safely used if the input dataset has no nodata value and *--nodata* is not specified.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the GDALG output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

--nodata <NODATA>

Override nodata value of input band(s) (numeric value, 'nan', 'inf', '-inf'). Either one value can be specified or as many values as they are input bands. When multiple values are specified, a pixel is considered as invalid only if it matches the nodata values on all bands.

Standard Options

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>

Output data type among Byte, Int8, UInt16, Int16, UInt32, Int32, UInt64, Int64, CInt16, CInt32, Float32, Float64, CFloat32, CFloat64.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1

```
$ gdal raster nodata-to-alpha input_with_nodata.tif output_with_alpha_band.tif
```

5.1.3.23 gdal raster overview

Added in version 3.11.

Synopsis

```
Usage: gdal raster overview <SUBCOMMAND> [OPTIONS]
where <SUBCOMMAND> is one of:
  - add:     Adding overviews.
  - delete:  Deleting overviews.
  - refresh: Refresh overviews.

Common Options:
  -h, --help              Display help message and exit
  --json-usage            Display usage as JSON document and exit
  --config <KEY>=<VALUE>  Configuration option [may be repeated]
```

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Available sub-commands

- *gdal raster overview add*
- *gdal raster overview delete*
- *gdal raster overview refresh*

5.1.3.24 gdal raster overview add

Added in version 3.11.

Synopsis

```
Usage: gdal raster overview add [OPTIONS] <INPUT>

Adding overviews.

Positional arguments:
  -i, --dataset, --input <INPUT>         Dataset (to be updated in-place, unless --
↪external) [required]

Common Options:
  -h, --help                             Display help message and exit
  --json-usage                           Display usage as JSON document and exit
  --config <KEY>=<VALUE>                 Configuration option [may be repeated]
  -q, --quiet                            Quiet mode (no progress bar or warning message)␣
↪[not available in pipelines]

Options:
  --overview-src <OVERVIEW-SRC>          Source overview dataset [may be repeated]
                                         Mutually exclusive with --levels
  --external                             Add external overviews
  -r, --resampling <RESAMPLING>          Resampling method.␣
↪RESAMPLING=nearest|average|cubic|cubicspline|lanczos|bilinear|gauss|average_
↪magphase|rms|mode
  --levels <LEVELS>                      Levels / decimation factors [may be repeated]
                                         Mutually exclusive with --overview-src
  --min-size <MIN-SIZE>                  Maximum width or height of the smallest␣
↪overview level.
  --co, --creation-option <KEY>=<VALUE>  Overview creation option [may be repeated]

Advanced Options:
  --oo, --open-option <KEY>=<VALUE>      Open options [may be repeated]
```

Description

gdal raster overview add can be used to build or rebuild overview images for most supported file formats with one of several downsampling algorithms.

Starting with GDAL 3.12, **gdal raster overview add** can be used as a step of a pipeline. In that case virtual overviews are generated at the output of the step, and can potentially be materialized in the final write step, when writing a GeoTIFF file with the COPY_SRC_OVERVIEWS creation option, or when writing to a COG (Cloud Optimized GeoTIFF) file.

Program-Specific Options

--co <NAME>=<VALUE>

Added in version 3.12.

Overview creation options. May be repeated.

Many formats have one or more optional creation options that can be used to control particulars about the created overviews. Options available can be obtained by looking at the `OverviewCreationOptionList` returned by `gdal --format <FORMAT-NAME>`.

Most formats will support external overviews in a GeoTIFF file in a side-car file of extension `.ovr`. You can consult the *overview creation options for GeoTIFF*.

--dataset <DATASET>

Dataset name, to be updated in-place by default (unless *--external* is specified). Required for standalone execution, implicit when run as a pipeline step

--external

Create external `.ovr` overviews as GeoTIFF files. Not available when run as a pipeline step.

--levels <level1,level2,...>

A list of overview levels to build. Each overview level must be an integer value greater or equal to 2.

When explicit levels are not specified,

- If there are already existing overviews, the corresponding levels will be used to refresh them if no explicit levels are specified.
- Otherwise, appropriate overview power-of-two factors will be selected until the smallest overview is smaller than the value of the *--min-size* switch.

Mutually exclusive with *--overview-src*

--min-size <val>

Maximum width or height of the smallest overview level. Only taken into account if explicit levels are not specified. Defaults to 256.

--overview-src <INPUT>

Added in version 3.12.

Add specified input raster datasets as overviews of the target dataset. Source overviews may come from any GDAL supported format, provided they have the same number of bands and geospatial extent than the target dataset.

That mode is currently only implemented when the target dataset is in GeoTIFF format, or when using *--external*.

Mutually exclusive with *--levels*

-r,
--resampling {nearest|average|cubic|cubicspline|lanczos|bilinear|gauss|average_magphase|rms|mode}

Select a resampling algorithm. The default is `nearest`, which is generally not appropriate if sub-pixel accuracy is desired.

When refreshing existing TIFF overviews, the previously used method, as noted in the RESAMPLING metadata item of the overview, will be used if *-r* is not specified.

The available methods are:

`nearest` applies a nearest neighbour (simple sampling) resampler.

`average` computes the average of all non-NODATA contributing pixels. This is a weighted average taking into account properly the weight of source pixels not contributing fully to the target pixel.

`bilinear` applies a bilinear convolution kernel.

`cubic` applies a cubic convolution kernel.

`cubicspline` applies a B-Spline convolution kernel.

`lanczos` applies a Lanczos windowed sinc convolution kernel.

`gauss` applies a Gaussian kernel before computing the overview, which can lead to better results than simple averaging in e.g case of sharp edges with high contrast or noisy patterns. The advised level values should be 2, 4, 8, ... so that a 3x3 resampling Gaussian kernel is selected.

`average_magphase` averages complex data in mag/phase space.

`rms` computes the root mean squared / quadratic mean of all non-NODATA contributing pixels

`mode` selects the value which appears most often of all the sampled points.

Standard Options

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Create overviews, embedded in the supplied TIFF file, with automatic computation of levels

```
gdal raster overview add -r average abc.tif
```

Example 2: Create overviews, embedded in the supplied TIFF file

```
gdal raster overview add -r average --levels=2,4,8,16 abc.tif
```

Example 3: Create an external compressed GeoTIFF overview file from the ERDAS .IMG file

```
gdal raster overview add --external --levels=2,4,8,16 --co COMPRESS=YES erdas.img
```

Example 4: Create an external JPEG-compressed GeoTIFF overview file from a 3-band RGB dataset

If the dataset is a writable GeoTIFF, you also need to add the `--external` option to force the generation of external overview.

```
gdal raster overview add --co OVERVIEW=JPEG --co PHOTOMETRIC=YCBCR \
                         --co INTERLEAVE=PIXEL rgb_dataset.ext 2 4 8 16
```

Example 5: Create overviews for a specific subdataset

For example, one of potentially many raster layers in a GeoPackage (the "filename" parameter must be driver prefix, filename and subdataset name, like e.g. shown by gdalinfo):

```
gdal raster overview add GPKG:file.gpkg:layer
```

Example 6: Add 3 existing datasets at scale 1:25K, 1:50K and 1:100K as overviews of my.tif.

```
gdal raster overview add --overview-src ovr_25k.tif --overview-src ovr_50k.tif --
↪overview-src ovr_100k.tif --dataset my.tif
```

Example 7: Create a COG file with non power-of-two overview levels.

```
gdal pipeline read input.tif ! reproject --output-crs=EPSG:4326 ! add overview --levels␣
↪16,64,128 ! write output.tif --format=COG
```

5.1.3.25 gdal raster overview delete

Added in version 3.11.

Synopsis

```
Usage: gdal raster overview delete [OPTIONS] <INPUT>

Deleting overviews.

Positional arguments:
  -i, --dataset, --input <INPUT>     Dataset (to be updated in-place, unless --external)␣
↪[required]

Common Options:
  -h, --help                         Display help message and exit
  --json-usage                       Display usage as JSON document and exit
  --config <KEY>=<VALUE>             Configuration option [may be repeated]
  -q, --quiet                        Quiet mode (no progress bar or warning message)␣
↪[not available in pipelines]

Options:
  --external                         Delete external overviews

Advanced Options:
  --oo, --open-option <KEY>=<VALUE>  Open options [may be repeated]
```

Description

gdal raster overview delete can be used to delete all existing overviews of a dataset.

> **Note**

For most file formats (including GeoTIFF or GeoPackage), the space previously occupied by the removed overviews may not be reclaimed. It might be needed for use *gdal raster convert* to create a new compact dataset.

Program-Specific Options

--dataset <DATASET>

Dataset name, to be updated in-place by default (unless `--external` is specified). Required.

--external

Remove external `.ovr` overviews.

Standard Options

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Delete overviews of a GeoTIFF file.

```
gdal raster overview delete my.tif
```

5.1.3.26 gdal raster overview refresh

Added in version 3.12.

Synopsis

```
Usage: gdal raster overview refresh [OPTIONS] <INPUT>

Refresh overviews.

Positional arguments:
  -i, --dataset, --input <INPUT>     Dataset (to be updated in-place, unless --external)␣
↪[required]

Common Options:
  -h, --help                         Display help message and exit
  --json-usage                       Display usage as JSON document and exit
  --config <KEY>=<VALUE>             Configuration option [may be repeated]
  -q, --quiet                        Quiet mode (no progress bar or warning message)␣
↪[not available in pipelines]

Options:
  --external                         Refresh external overviews
```

(continues on next page)

(continued from previous page)

```
  -r, --resampling <RESAMPLING>      Resampling method.␣
↪RESAMPLING=nearest|average|cubic|cubicspline|lanczos|bilinear|gauss|average_
↪magphase|rms|mode
  --levels <LEVELS>                  Levels / decimation factors [may be repeated]
  --bbox <BBOX>                      Bounding box to refresh
                                     Mutually exclusive with --like, --use-source-
↪timestamp
  --like <LIKE>                      Use extent of dataset(s) [may be repeated]
                                     Mutually exclusive with --bbox, --use-source-
↪timestamp
  --use-source-timestamp             Use timestamp of VRT or GTI sources as refresh␣
↪criterion
                                     Mutually exclusive with --bbox, --like

Advanced Options:
  --oo, --open-option <KEY>=<VALUE>  Open options [may be repeated]
```

Description

gdal raster overview refresh can be used to refresh existing overviews of a dataset. By default all overviews are refreshed, but it is also possible to restrict the refreshed overviews by level and/or extent.

Program-Specific Options

--bbox `<xmin>,<ymin>,<xmax>,ymax>`

This option performs a partial refresh of existing overviews, in the region of interest specified by georeferenced coordinates, in CRS units.

'x' is longitude values for geographic CRS and easting for projected CRS. 'y' is latitude values for geographic CRS and northing for projected CRS.

--dataset `<DATASET>`

Dataset name, to be updated in-place by default (unless *--external* is specified). Required.

--external

Refresh external `.ovr` overviews.

--levels `<level1,level2,...>`

A list of overview levels to build. Each overview level must be an integer value greater or equal to 2.

If not specified all existing overviews are refreshed.

--like `<filename1>[,<filenameN>]...`

This option performs a partial refresh of existing overviews, in the region of interest specified by one or several filenames (names separated by comma). Note that the filenames are only used to determine the regions of interest to refresh. The reference source pixels are the one of the main dataset. By default all existing overview levels will be refreshed, unless explicit levels are specified. See *Example 2*.

-r,
--resampling `{nearest|average|cubic|cubicspline|lanczos|bilinear|gauss|average_magphase|rms|mode}`

Select a resampling algorithm. The default is `nearest`, which is generally not appropriate if sub-pixel accuracy is desired.

When refreshing existing TIFF overviews, the previously used method, as noted in the RESAMPLING metadata item of the overview, will be used if *-r* is not specified.

The available methods are:

`nearest` applies a nearest neighbour (simple sampling) resampler.

`average` computes the average of all non-NODATA contributing pixels. This is a weighted average taking into account properly the weight of source pixels not contributing fully to the target pixel.

`bilinear` applies a bilinear convolution kernel.

`cubic` applies a cubic convolution kernel.

`cubicspline` applies a B-Spline convolution kernel.

`lanczos` applies a Lanczos windowed sinc convolution kernel.

`gauss` applies a Gaussian kernel before computing the overview, which can lead to better results than simple averaging in e.g case of sharp edges with high contrast or noisy patterns. The advised level values should be 2, 4, 8, ... so that a 3x3 resampling Gaussian kernel is selected.

`average_magphase` averages complex data in mag/phase space.

`rms` computes the root mean squared / quadratic mean of all non-NODATA contributing pixels

`mode` selects the value which appears most often of all the sampled points.

--use-source-timestamp

This option performs a partial refresh of existing overviews of a *VRT -- GDAL Virtual Format* or *GTI -- GDAL Raster Tile Index* file with an external overview. It checks the modification timestamp of all the sources of the VRT and regenerate the overview for areas corresponding to sources whose timestamp is more recent than the external overview of the VRT. By default all existing overview levels will be refreshed, unless explicit levels are specified. See *Example 1*.

Standard Options

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Refresh external overviews of a VRT file using timestamp of source files

This is needed when for sources have been modified after the .vrt.ovr generation:

```
gdal raster mosaic tile1.tif tile2.tif my.vrt              # create VRT
gdal raster overview add --external -r cubic my.vrt        # initial overview generation
touch tile1.tif                                            # simulate update of one of␣
↪the source tiles
gdal raster overview refresh --external -r cubic \
                          --use-source-timestamp my.vrt    # refresh overviews
```

Example 2: Refresh (internal) overviews of a TIFF file

```
gdal raster mosaic tile1.tif tile2.tif mosaic.tif       # create mosaic
gdal raster overview add -r cubic mosaic.tif            # initial overview generation
gdalwarp tile1_modif.tif mosaic.tif                     # update mosaic
gdal raster overview refresh --like=tile1.tif my.tif    # refresh overviews
```

5.1.3.27 gdal raster pansharpen

Added in version 3.12.

Synopsis

```
Usage: gdal raster pansharpen [OPTIONS] <INPUT> <SPECTRAL>... <OUTPUT>

Perform a pansharpen operation.

Positional arguments:
  -i, --panchromatic, --input <INPUT>                      Input panchromatic raster␣
↪dataset [required] [not available in pipelines]
  --spectral <SPECTRAL>                                    Input spectral band dataset␣
↪[1.. values] [required]
  -o, --output <OUTPUT>                                    Output raster dataset␣
↪[required] [not available in pipelines]

Common Options:
  -h, --help                                               Display help message and exit
  --json-usage                                             Display usage as JSON␣
↪document and exit
  --config <KEY>=<VALUE>                                   Configuration option [may be␣
↪repeated]
  -q, --quiet                                              Quiet mode (no progress bar␣
↪or warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>      Output format ("GDALG"␣
↪allowed) [not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                    Creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                              Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                           Mutually exclusive with --
↪append
  --append                                                 Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                           Mutually exclusive with --
↪overwrite
  -r, --resampling <RESAMPLING>                            Resampling algorithm.␣
↪RESAMPLING=nearest|bilinear|cubic|cubicspline|lanczos|average (default: cubic)
  --weights <WEIGHTS>                                      Weight for each input␣
↪spectral band [may be repeated]
  --nodata <NODATA>                                        Override nodata value of␣
↪input bands
```

(continues on next page)

(continued from previous page)

```
  --bit-depth <BIT-DEPTH>                                  Override bit depth of input␣
↪bands
  --spatial-extent-adjustment <SPATIAL-EXTENT-ADJUSTMENT>  Select behavior when bands␣
↪have not the same extent. SPATIAL-EXTENT-ADJUSTMENT=union|intersection|none|none-
↪without-warning (default: union)
  -j, --num-threads <NUM-THREADS>                          Number of jobs (or ALL_CPUS)␣
↪(default: ALL_CPUS)

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                      Input formats [may be␣
↪repeated] [not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                        Open options [may be␣
↪repeated] [not available in pipelines]
```

Description

gdal raster pansharpen performs a pan-sharpening operation. It can create a "classic" output dataset (such as GeoTIFF), or a VRT dataset describing the pan-sharpening operation.

It takes as input a one-band panchromatic dataset and several spectral bands, generally at a lower resolution than the panchromatic band. The output is a multi-band dataset (with as many bands as input panchromatic bands), enhanced at the resolution of the panchromatic band.

All bands should be in the same coordinate reference system.

More details can be found in the *Pansharpened VRT* section.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

--bit-depth <val>

Specify the bit depth of the panchromatic and spectral bands (e.g. 12). If not specified, the NBITS metadata item from the panchromatic band will be used if it exists.

-i, --panchromatic, --input <INPUT>

Dataset with panchromatic band. [required]

-j, --num-threads <value>

Specify number of threads to use to do the resampling and pan-sharpening itself. Can be an integer number or `ALL_CPUS` (the default)

--nodata <val>

Specify nodata value for bands. Used for the resampling and pan-sharpening computation itself. If not set, deduced from the input bands, provided they have a consistent setting.

-r, --resampling `nearest|bilinear|cubic|cubicspline|lanczos|average`

Select a resampling algorithm. `cubic` is the default.

--spatial-extent-adjustment union|intersection|none|none-without-warning

Select behavior when bands have not the same extent. See *SpatialExtentAdjustment* documentation in *Pansharpened VRT* union is the default.

--spectral <spectral_dataset>[,band=<num>]

Dataset with one or several spectral bands. [required]

If the band option is not specified, all bands of the dataset are taken into account. Otherwise, only the specified (num)th band. The same dataset can be repeated several times.

--weights <WEIGHTS>

Specify a weight for the computation of the pseudo panchromatic value. There must be as many values as input spectral bands.

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: With spectral bands in a single dataset

```
gdal raster pansharpen panchro.tif rgb.tif pansharpened_out.tif
```

Example 2: With a few spectral bands from a single dataset, reordered

```
gdal raster pansharpen panchro.tif bgr.tif,band=3 bgr.tif,band=2 bgr.tif,band=1␣
↪pansharpened_out.tif
```

Example 3: With spectral bands in several datasets

```
gdal raster pansharpen panchro.tif red.tif green.tif blue.tif pansharpened_out.tif
```

Example 4: Specifying weights

```
gdal raster pansharpen -w 0.7,0.2,0.1 panchro.tif multispectral.tif pansharpened_out.tif
```

Example 5: Select RGB bands from a RGBNir multispectral dataset while computing the pseudo panchromatic intensity on the 4 RGBNir bands

```
gdal raster pipeline read panchro.tif ! pansharpen rgbnir.tif ! select 1,2,3 ! write␣
↪pansharpened_out.tif
```

5.1.3.28 `gdal raster pipeline`

Added in version 3.11.

Description

`gdal raster pipeline` can be used to process a raster dataset and perform various processing steps that accept raster and generate raster.

For pipelines mixing raster and vector, consult *gdal pipeline*.

Most steps proceed in on-demand evaluation of raster blocks, unless otherwise stated in their documentation, without "materializing" the resulting dataset of the operation of each step. It may be desirable sometimes for performance purposes to proceed to materializing an intermediate dataset to disk using *gdal raster materialize*.

Synopsis

```
Usage: gdal raster pipeline [OPTIONS] <PIPELINE>

Process a raster dataset applying several steps.
```

(continues on next page)

(continued from previous page)

```
Positional arguments:

Common Options:
  -h, --help              Display help message and exit
  --json-usage            Display usage as JSON document and exit
  --config <KEY>=<VALUE>  Configuration option [may be repeated]
  -q, --quiet             Quiet mode (no progress bar or warning message) [not available␣
↪in pipelines]

<PIPELINE> is of the form: read|mosaic|stack [READ-OPTIONS] ( ! <STEP-NAME> [STEP-
↪OPTIONS] )* ! info|compare|tile|write [WRITE-OPTIONS]
```

A pipeline chains several steps, separated with the *!* (exclamation mark) character. The first step must be `read`, `calc`, `create`, `mosaic` or `stack`, and the last one `write`, `info` or `tile`. Each step has its own positional or non-positional arguments. Apart from `read`, `calc`, `mosaic`, `stack`, `compare`, `info`, `tile` and `write`, all other steps can potentially be used several times in a pipeline.

Steps

- *aspect*
- *blend*
- *calc*
- *clip*
- *color-map*
- *compare*
- *create*
- *edit*
- *external*
- *fill-nodata*
- *hillshade*
- *info*
- *materialize*
- *mosaic*
- *neighbors*
- *nodata-to-alpha*
- *overview*
- *pansharpen*
- *proximity*
- *read*
- *reclassify*

- *reproject*
- *resize*
- *rgb-to-palette*
- *roughness*
- *scale*
- *select*
- *set-type*
- *sieve*
- *slope*
- *stack*
- *tee*
- *tile*
- *tpi*
- *tri*
- *unscale*
- *update*
- *viewshed*
- *write*

aspect

```
* aspect [OPTIONS]
------------------

Generate an aspect map

Options:
  -b, --band <BAND>              Input band (1-based index) (default: 1)
  --convention <CONVENTION>      Convention for output angles.␣
↪CONVENTION=azimuth|trigonometric-angle (default: azimuth)
  --gradient-alg <GRADIENT-ALG>  Algorithm used to compute terrain gradient. GRADIENT-
↪ALG=Horn|ZevenbergenThorne (default: Horn)
  --zero-for-flat                Whether to output zero for flat areas
  --no-edges                     Do not try to interpolate values at dataset edges or␣
↪close to nodata values
```

Details for options can be found in *gdal raster aspect*.

blend

```
* blend [OPTIONS] <OVERLAY>
---------------------------
```

(continues on next page)

(continued from previous page)

```
Blend/compose two raster datasets

Positional arguments:
  --overlay <OVERLAY>     Overlay dataset [required]

Options:
  --operator <OPERATOR>   Composition operator. OPERATOR=src-over|hsv-
↪value|multiply|screen|overlay|hard-light|darken|lighten|color-dodge|color-burn␣
↪(default: src-over)
  --opacity <OPACITY>     Opacity percentage to apply to the overlay dataset (0=fully␣
↪transparent, 100=full use of overlay opacity) (default: 100)
```

Details for options can be found in *gdal raster blend*.

calc

```
* calc [OPTIONS] <INPUTS>...
----------------------------

Perform raster algebra

Positional arguments:
  -i, --input <INPUTS>                                    Input raster datasets [1..␣
↪values] [required]

Options:
  --ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>  Output data type. OUTPUT-DATA-
↪TYPE=UInt8|Int8|UInt16|Int16|UInt32|Int32|UInt64|Int64|CInt16|CInt32|Float16|Float32|Float64|CFloat32
  --no-check-crs                                          Do not check consistency of␣
↪input coordinate reference systems
  --no-check-extent                                       Do not check consistency of␣
↪input extents
  --propagate-nodata                                      Whether to set pixels to the␣
↪output NoData value if any of the input pixels is NoData
  --calc <CALC>                                           Expression(s) to evaluate [1..
↪ values] [required]
  --dialect <DIALECT>                                     Expression dialect.␣
↪DIALECT=muparser|builtin (default: muparser)
  --flatten                                               Generate a single band output␣
↪raster per expression, even if input datasets are multiband
  --nodata <NODATA>                                       Assign a specified nodata␣
↪value to output bands ('none', numeric value, 'nan', 'inf', '-inf')

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                     Input formats [may be␣
↪repeated]
  --oo, --open-option <KEY>=<VALUE>                       Open options [may be repeated]
```

Details for options can be found in *gdal raster calc*.

clip

```
* clip [OPTIONS]
----------------

Clip a raster dataset.

Options:
  --bbox <BBOX>                    Clipping bounding box as xmin,ymin,xmax,ymax
                                   Mutually exclusive with --window, --geometry, --like
  --bbox-crs <BBOX-CRS>            CRS of clipping bounding box
  --window <WINDOW>                Raster window as col,line,width,height in pixels
                                   Mutually exclusive with --bbox, --geometry, --like
  --geometry <GEOMETRY>            Clipping geometry (WKT or GeoJSON)
                                   Mutually exclusive with --bbox, --window, --like
  --geometry-crs <GEOMETRY-CRS>    CRS of clipping geometry
  --like <DATASET>                 Dataset to use as a template for bounds
                                   Mutually exclusive with --bbox, --window, --geometry
  --like-sql <SELECT-STATEMENT>    SELECT statement to run on the 'like' dataset
                                   Mutually exclusive with --like-where
  --like-layer <LAYER-NAME>        Name of the layer of the 'like' dataset
  --like-where <WHERE-EXPRESSION>  WHERE SQL clause to run on the 'like' dataset
                                   Mutually exclusive with --like-sql
  --only-bbox                      For 'geometry' and 'like', only consider their␣
↪bounding box
  --allow-bbox-outside-source      Allow clipping box to include pixels outside input␣
↪dataset
  --add-alpha                      Adds an alpha mask band to the destination when the␣
↪source raster have none.
```

Details for options can be found in *gdal raster clip*.

color-map

```
* color-map [OPTIONS]
---------------------

Generate a RGB or RGBA dataset from a single band, using a color map

Options:
  -b, --band <BAND>                    Input band (1-based index) (default: 1)
  --color-map <COLOR-MAP>              Color map filename
  --add-alpha                          Adds an alpha mask band to the destination.
  --color-selection <COLOR-SELECTION>  How to compute output colors from input values.␣
↪COLOR-SELECTION=interpolate|exact|nearest (default: interpolate)
```

Details for options can be found in *gdal raster color-map*.

compare

Added in version 3.12.

```
* compare [OPTIONS] <REFERENCE>
------------------------------

Compare two raster datasets.

Positional arguments:
  --reference <REFERENCE>  Reference dataset [required]

Options:
  --skip-all-optional      Skip all optional comparisons
  --skip-binary            Skip binary file comparison
  --skip-crs               Skip CRS comparison
  --skip-geotransform      Skip geotransform comparison
  --skip-overview          Skip overview comparison
  --skip-metadata          Skip metadata comparison
  --skip-rpc               Skip RPC metadata comparison
  --skip-geolocation       Skip Geolocation metadata comparison
  --skip-subdataset        Skip subdataset comparison
```

Details for options can be found in *gdal raster compare*.

create

```
* create [OPTIONS]
------------------

Create a new raster dataset.

Options:
  -i, --like, --input <TEMPLATE-DATASET>                   Template raster dataset
  --size <width>,<height>                                   Output size in pixels
  --band-count <BAND-COUNT>                                 Number of bands (default: 1)
  --ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>  Output data type. OUTPUT-DATA-
↪TYPE=UInt8|Int8|UInt16|Int16|UInt32|Int32|UInt64|Int64|CInt16|CInt32|Float16|Float32|Float64|CFloat32
↪(default: Byte)
  --nodata <NODATA>                                         Assign a specified nodata␣
↪value to output bands ('none', numeric value, 'nan', 'inf', '-inf')
  --burn <BURN>                                             Burn value [may be repeated]
  --crs <CRS>                                               Set CRS
  --bbox <BBOX>                                             Bounding box as xmin,ymin,
↪xmax,ymax
  --metadata <KEY>=<VALUE>                                  Add metadata item [may be␣
↪repeated]
  --copy-metadata                                           Copy metadata from input␣
↪dataset
                                                            Depends on --input
  --copy-overviews                                          Create same overview levels␣
↪as input dataset
                                                            Depends on --input

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                       Input formats [may be␣
```

(continues on next page)

(continued from previous page)

```
↪repeated]
  --oo, --open-option <KEY>=<VALUE>                        Open options [may be repeated]
```

Details for options can be found in *gdal raster create*.

edit

```
* edit [OPTIONS]
----------------

Edit a raster dataset.

Options:
  -b, --band <BAND>                                        Active band (1-based index)
  --crs <CRS>                                              Override CRS (without␣
↪reprojection)
  --bbox <BBOX>                                            Bounding box as xmin,ymin,
↪xmax,ymax
  --nodata <NODATA>                                        Assign a specified nodata␣
↪value to output bands ('none', numeric value, 'nan', 'inf', '-inf')
  --color-interpretation <[all|<BAND>=]<COLOR-INTEPRETATION>  Set band color␣
↪interpretation [may be repeated]
  --color-map <COLOR-MAP>                                  Color map filename
                                                           Mutually exclusive with --
↪unset-color-table
  --scale <[<BAND>=]<SCALE>                                Override band scale factor␣
↪[may be repeated]
  --offset <[<BAND>=]<OFFSET>                              Override band offset␣
↪constant [may be repeated]
  --metadata <KEY>=<VALUE>                                 Add/update dataset␣
↪metadata item [may be repeated]
  --unset-color-table                                      Unset color table on 1st␣
↪band or the one specified with --band
                                                           Mutually exclusive with --
↪color-map
  --unset-metadata <KEY>                                   Remove dataset metadata␣
↪item(s) [may be repeated]
  --unset-metadata-domain <DOMAIN>                         Remove dataset metadata␣
↪domain(s) [may be repeated]
  --gcp <GCP>                                              Add ground control point,␣
↪formatted as pixel,line,easting,northing[,elevation], or @filename [may be repeated]
```

Details for options can be found in *gdal raster edit*.

external

```
* external [OPTIONS] <COMMAND>
------------------------------

Execute an external program as a step of a pipeline
```

(continues on next page)

(continued from previous page)

```
Positional arguments:
  --command <COMMAND>                                External command, optionally with
↪<INPUT> and/or <OUTPUT> or <INPUT-OUTPUT> placeholders [required]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                  Input formats
```

Details for options can be found in *gdal external*.

fill-nodata

```
* fill-nodata [OPTIONS]
-----------------------

Fill nodata raster regions by interpolation from edges.

Options:
  -b, --band <BAND>                                  Input band (1-based index)␣
↪(default: 1)
  -d, --max-distance <MAX_DISTANCE>                  The maximum distance (in pixels)␣
↪that the algorithm will search out for values to interpolate. (default: 100)
  -s, --smoothing-iterations <SMOOTHING_ITERATIONS>  The number of 3x3 average filter␣
↪smoothing iterations to run after the interpolation to dampen artifacts. The default␣
↪is zero smoothing iterations. (default: 0)
  --mask <MASK>                                      Use the first band of the specified␣
↪file as a validity mask (zero is invalid, non-zero is valid).
  --strategy <STRATEGY>                              By default, pixels are interpolated␣
↪using an inverse distance weighting (invdist). It is also possible to choose a nearest␣
↪neighbour (nearest) strategy.. STRATEGY=invdist|nearest (default: invdist)
```

Details for options can be found in *gdal raster fill-nodata*.

hillshade

```
* hillshade [OPTIONS]
---------------------

Generate a shaded relief map

Options:
  -b, --band <BAND>              Input band (1-based index) (default: 1)
  -z, --zfactor <ZFACTOR>        Vertical exaggeration used to pre-multiply the␣
↪elevations
  --xscale <XSCALE>              Ratio of vertical units to horizontal X axis units
  --yscale <YSCALE>              Ratio of vertical units to horizontal Y axis units
  --azimuth <AZIMUTH>            Azimuth of the light, in degrees (default: 315)
  --altitude <ALTITUDE>          Altitude of the light, in degrees (default: 45)
  --gradient-alg <GRADIENT-ALG>  Algorithm used to compute terrain gradient. GRADIENT-
```

(continues on next page)

(continued from previous page)

```
↪ALG=Horn|ZevenbergenThorne (default: Horn)
  --variant <VARIANT>            Variant of the hillshading algorithm.␣
↪VARIANT=regular|combined|multidirectional|Igor (default: regular)
  --no-edges                     Do not try to interpolate values at dataset edges or␣
↪close to nodata values
```

Details for options can be found in *gdal raster hillshade*.

info

Added in version 3.12.

```
* info [OPTIONS]
----------------

Return information on a raster dataset.

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format. OUTPUT-
↪FORMAT=json|text
  --mm, --min-max                                      Compute minimum and maximum value
  --stats                                              Retrieve or compute statistics,␣
↪using all pixels
                                                       Mutually exclusive with --approx-
↪stats
  --approx-stats                                       Retrieve or compute statistics,␣
↪using a subset of pixels
                                                       Mutually exclusive with --stats
  --hist                                               Retrieve or compute histogram

Advanced Options:
  --no-gcp                                             Suppress ground control points␣
↪list printing
  --no-md                                              Suppress metadata printing
  --no-ct                                              Suppress color table printing
  --no-fl                                              Suppress file list printing
  --checksum                                           Compute pixel checksum
  --list-metadata-domains, --list-mdd                  List all metadata domains␣
↪available for the dataset
  --mdd, --metadata-domain <METADATA-DOMAIN>           Report metadata for the specified␣
↪domain. 'all' can be used to report metadata in all domains

Esoteric Options:
  --no-nodata                                          Suppress retrieving nodata value
  --no-mask                                            Suppress mask band information
  --subdataset <SUBDATASET>                            Use subdataset of specified index␣
↪(starting at 1), instead of the source dataset itself
  --crs-format <CRS-FORMAT>                            Which format to use to report CRS.
↪ CRS-FORMAT=AUTO|WKT2|PROJJSON (default: AUTO)
```

Details for options can be found in *gdal raster info*.

materialize

```
* materialize [OPTIONS]
-----------------------

Materialize a piped dataset on disk to increase the efficiency of the following steps.

Options:
  -o, --output <OUTPUT>                                Materialized dataset name␣
↪(created by algorithm)
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format
  --co, --creation-option <KEY>=<VALUE>                Creation option [may be repeated]
  --overwrite                                          Whether overwriting existing␣
↪output dataset is allowed
```

Details for options can be found in *gdal raster materialize*.

mosaic

```
* mosaic [OPTIONS] <INPUTS>...
------------------------------

Build a mosaic, either virtual (VRT) or materialized.

Positional arguments:
  -i, --input <INPUTS>                                          Input raster datasets␣
↪(or specify a @<filename> to point to a file containing filenames) [1.. values]␣
↪[required]

Options:
  -b, --band <BAND>                                             Input band(s) (1-based␣
↪index) [may be repeated]
  --absolute-path                                               Whether the path to␣
↪the input datasets should be stored as an absolute path
  --resolution <xres>,<yres>|same|average|common|highest|lowest>  Target resolution (in␣
↪destination CRS units) (default: same)
  --bbox <BBOX>                                                 Target bounding box as␣
↪xmin,ymin,xmax,ymax (in destination CRS units)
  --target-aligned-pixels                                       Round target extent to␣
↪target resolution
  --input-nodata <INPUT-NODATA>                                 Set nodata values for␣
↪input bands. [1.. values]
  --output-nodata <OUTPUT-NODATA>                               Set nodata values at␣
↪the destination band level. [1.. values]
  --hide-nodata                                                 Makes the destination␣
↪band not report the NoData.
  --add-alpha                                                   Adds an alpha mask␣
↪band to the destination when the source raster have none.
  --pixel-function <PIXEL-FUNCTION>                             Specify a pixel␣
↪function to calculate output value from overlapping inputs. PIXEL-
↪FUNCTION=abs|area|argmax|argmin|cmul|complex|conj|dB|dB2amp|dB2pow|diff|div|exp|expression|geometric_
↪mean|harmonic_mean|imag|intensity|interpolate_exp|interpolate_
↪linear|inv|log10|max|mean|median|min|mod|mode|mul|norm_
```

(continues on next page)

(continued from previous page)

```
↪diff|phase|polar|pow|quantile|real|reclassify|replace_nodata|round|scale|sqrt|sum
  --pixel-function-arg <NAME>=<VALUE>                              Specify argument(s) to␣
↪pass to the pixel function [may be repeated]

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                              Input formats [may be␣
↪repeated]
  --oo, --open-option <KEY>=<VALUE>                                Open options [may be␣
↪repeated]
```

Details for options can be found in *gdal raster mosaic*.

neighbors

```
* neighbors [OPTIONS]
---------------------

Compute the value of each pixel from its neighbors (focal statistics)

Options:
  -b, --band <BAND>                                        Input band (1-based index)
  --method <METHOD>                                        Method to combine weighed␣
↪source pixels. METHOD=mean|sum|min|max|stddev|median|mode [may be repeated]
  --size <SIZE>                                            Neighborhood size
  --kernel <KERNEL>                                        Convolution kernel(s) to␣
↪apply [1.. values] [required]
  --ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>  Output data type. OUTPUT-DATA-
↪TYPE=UInt8|Int8|UInt16|Int16|UInt32|Int32|UInt64|Int64|CInt16|CInt32|Float16|Float32|Float64|CFloat32
↪(default: Float64)
  --nodata <NODATA>                                        Assign a specified nodata␣
↪value to output bands ('none', numeric value, 'nan', 'inf', '-inf')
```

Details for options can be found in *gdal raster neighbors*.

nodata-to-alpha

```
* nodata-to-alpha [OPTIONS]
---------------------------

Replace nodata value(s) with an alpha band.

Options:
  --nodata <NODATA>       Override nodata value of input band(s) (numeric value, 'nan',
↪'inf', '-inf') [may be repeated]
```

Details for options can be found in *gdal raster nodata-to-alpha*.

overview

```
* overview <COMMAND> [OPTIONS]
------------------------------
```

(continues on next page)

(continued from previous page)

```
where <COMMAND> is one of:
  - add: Adding overviews.
```

Details for options can be found in *gdal raster overview*.

pansharpen

```
* pansharpen [OPTIONS] <SPECTRAL>...
-----------------------------------

Perform a pansharpen operation.

Positional arguments:
  --spectral <SPECTRAL>                                   Input spectral band dataset␣
↪[1.. values] [required]

Options:
  -r, --resampling <RESAMPLING>                           Resampling algorithm.␣
↪RESAMPLING=nearest|bilinear|cubic|cubicspline|lanczos|average (default: cubic)
  --weights <WEIGHTS>                                     Weight for each input␣
↪spectral band [may be repeated]
  --nodata <NODATA>                                       Override nodata value of␣
↪input bands
  --bit-depth <BIT-DEPTH>                                 Override bit depth of input␣
↪bands
  --spatial-extent-adjustment <SPATIAL-EXTENT-ADJUSTMENT>  Select behavior when bands␣
↪have not the same extent. SPATIAL-EXTENT-ADJUSTMENT=union|intersection|none|none-
↪without-warning (default: union)
  -j, --num-threads <NUM-THREADS>                         Number of jobs (or ALL_CPUS)␣
↪(default: ALL_CPUS)
```

Details for options can be found in *gdal raster pansharpen*.

proximity

```
* proximity [OPTIONS]
---------------------

Produces a raster proximity map.

Options:
  --ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>  Output data type. OUTPUT-DATA-
↪TYPE=UInt8|Int8|UInt16|Int16|UInt32|Int32|UInt64|Int64|CInt16|CInt32|Float16|Float32|Float64|CFloat32
↪(default: Float32)
  -b, --band <BAND>                                       Input band (1-based index)
  --target-values <TARGET-VALUES>                         Target pixel values [may be␣
↪repeated]
  --distance-units <DISTANCE-UNITS>                       Distance units. DISTANCE-
↪UNITS=pixel|geo (default: pixel)
  --max-distance <MAX-DISTANCE>                           Maximum distance. The nodata␣
```

(continues on next page)

(continued from previous page)

```
↪value will be used for pixels beyond this distance (default: 0)
  --fixed-value <FIXED-VALUE>                              Fixed value for the pixels␣
↪that are within the maximum distance (instead of the actual distance) (default: 0)
  --nodata <NODATA>                                        Specify a nodata value to use␣
↪for pixels that are beyond the maximum distance
```

Details for options can be found in *gdal raster proximity*.

read

```
* read [OPTIONS] <INPUT>
------------------------

Read a raster dataset.

Positional arguments:
  -i, --input <INPUT>                  Input raster datasets [required]

Advanced Options:
  --if, --input-format <INPUT-FORMAT>  Input formats [may be repeated]
  --oo, --open-option <KEY>=<VALUE>    Open options [may be repeated]
```

Details for options can be found in *gdal raster pipeline read*.

reclassify

```
* reclassify [OPTIONS]
----------------------

Reclassify values in a raster dataset

Options:
  -m, --mapping <MAPPING>                                  Reclassification mappings (or␣
↪specify a @<filename> to point to a file containing mappings [required]
  --ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>  Output data type. OUTPUT-DATA-
↪TYPE=UInt8|Int8|UInt16|Int16|UInt32|Int32|UInt64|Int64|CInt16|CInt32|Float16|Float32|Float64|CFloat32
```

Details for options can be found in *gdal raster reclassify*.

reproject

```
* reproject [OPTIONS]
---------------------

Reproject a raster dataset.

Options:
  -s, --input-crs <INPUT-CRS>                Input CRS
  --like <DATASET>                           Dataset to use as a template for target␣
↪bounds, CRS, size and nodata
  -d, --output-crs <OUTPUT-CRS>              Output CRS
```

(continues on next page)

(continued from previous page)

```
  -r, --resampling <RESAMPLING>              Resampling method.␣
↪RESAMPLING=nearest|bilinear|cubic|cubicspline|lanczos|average|rms|mode|min|max|med|q1|q3|sum␣
↪(default: nearest)
  --resolution <xres>,<yres>                 Target resolution (in destination CRS units)
                                             Mutually exclusive with --size
  --size <width>,<height>                    Target size in pixels
                                             Mutually exclusive with --resolution
  --bbox <BBOX>                              Target bounding box (in destination CRS␣
↪units)
  --bbox-crs <BBOX-CRS>                      CRS of target bounding box
  -j, --num-threads <NUM-THREADS>            Number of jobs (or ALL_CPUS) (default: ALL_
↪CPUS)

Advanced Options:
  --target-aligned-pixels                    Round target extent to target resolution
  --input-nodata <INPUT-NODATA>              Set nodata values for input bands ('None'␣
↪to unset). [1.. values]
  --output-nodata <OUTPUT-NODATA>            Set nodata values for output bands ('None'␣
↪to unset). [1.. values]
  --add-alpha                                Adds an alpha mask band to the destination␣
↪when the source raster have none.
  --wo, --warp-option <NAME>=<VALUE>         Warping option(s) [may be repeated]
  --to, --transform-option <NAME>=<VALUE>    Transform option(s) [may be repeated]
  --et, --error-threshold <ERROR-THRESHOLD>  Error threshold
```

Details for options can be found in *gdal raster reproject*.

resize

```
* resize [OPTIONS]
------------------

Resize a raster dataset without changing the georeferenced extents.

Options:
  --resolution <xres>,<yres>     Target resolution (in destination CRS units)
                                 Mutually exclusive with --size
  --size <width[%]>,<height[%]>  Target size in pixels (or percentage if using '%'␣
↪suffix) [required]
                                 Mutually exclusive with --resolution
  -r, --resampling <RESAMPLING>  Resampling method.␣
↪RESAMPLING=nearest|bilinear|cubic|cubicspline|lanczos|average|mode (default: nearest)
```

Details for options can be found in *gdal raster resize*.

rgb-to-palette

```
* rgb-to-palette [OPTIONS]
--------------------------

Convert a RGB image into a pseudo-color / paletted image.
```

(continues on next page)

(continued from previous page)

```
Options:
  --color-count <COLOR-COUNT>      Select the number of colors in the generated color␣
↪table (default: 256)
  --color-map <COLOR-MAP>          Color map filename
  --output-nodata <OUTPUT-NODATA>  Output nodata value
  --no-dither                      Disable Floyd-Steinberg dithering
  --bit-depth <BIT-DEPTH>          Bit depth of color palette component (8 bit causes␣
↪longer computation time). BIT-DEPTH=5|8 (default: 5)
```

Details for options can be found in *gdal raster rgb-to-palette*.

roughness

```
* roughness [OPTIONS]
---------------------

Generate a roughness map

Options:
  -b, --band <BAND>       Input band (1-based index) (default: 1)
  --no-edges              Do not try to interpolate values at dataset edges or close to␣
↪nodata values
```

Details for options can be found in *gdal raster roughness*.

scale

```
* scale [OPTIONS]
-----------------

Scale the values of the bands of a raster dataset.

Options:
  --ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>  Output data type. OUTPUT-DATA-
↪TYPE=UInt8|Int8|UInt16|Int16|UInt32|Int32|UInt64|Int64|CInt16|CInt32|Float16|Float32|Float64|CFloat32
  -b, --band <BAND>                                        Select band to restrict the␣
↪scaling (1-based index)
  --input-min <INPUT-MIN>                                  Minimum value of the source␣
↪range
                                                           Depends on --input-max
  --input-max <INPUT-MAX>                                  Maximum value of the source␣
↪range
                                                           Depends on --input-min
  --output-min <OUTPUT-MIN>                                Minimum value of the␣
↪destination range
                                                           Depends on --output-max
  --output-max <OUTPUT-MAX>                                Maximum value of the␣
↪destination range
                                                           Depends on --output-min
  --exponent <EXPONENT>                                    Exponent to apply non-linear␣
```

(continues on next page)

(continued from previous page)

```
↪scaling with a power function
  --no-clip                                                Do not clip input values to␣
↪[innput-min, input-max]
```

Details for options can be found in *gdal raster scale*.

select

```
* select [OPTIONS] <BAND>
-------------------------

Select a subset of bands from a raster dataset.

Positional arguments:
  -b, --band <BAND>       Band(s) (1-based index, 'mask', 'mask:<band>' or color␣
↪interpretation such as 'red') [1.. values] [required]

Options:
  --exclude               Exclude specified bands
  --mask <MASK>           Mask band (1-based index, 'mask', 'mask:<band>' or 'none')
```

Details for options can be found in *gdal raster select*.

set-type

```
* set-type [OPTIONS]
--------------------

Modify the data type of bands of a raster dataset.

Options:
  --ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>  Output data type. OUTPUT-DATA-
↪TYPE=UInt8|Int8|UInt16|Int16|UInt32|Int32|UInt64|Int64|CInt16|CInt32|Float16|Float32|Float64|CFloat32
↪[required]
```

Details for options can be found in *gdal raster set-type*.

sieve

```
* sieve [OPTIONS]
-----------------

Remove small polygons from a raster dataset.

Options:
  --mask <MASK>                          Use the first band of the specified file as a␣
↪validity mask (all pixels with a value other than zero will be considered suitable for␣
↪inclusion in polygons)
  -b, --band <BAND>                      Input band (1-based index)
  -s, --size-threshold <SIZE-THRESHOLD>  Minimum size of polygons to keep (default: 2)
  -c, --connect-diagonal-pixels          Consider diagonal pixels as connected
```

Details for options can be found in *gdal raster sieve*.

slope

```
* slope [OPTIONS]
-----------------

Generate a slope map

Options:
  -b, --band <BAND>              Input band (1-based index) (default: 1)
  --unit <UNIT>                  Unit in which to express slopes. UNIT=degree|percent␣
↪(default: degree)
  --xscale <XSCALE>              Ratio of vertical units to horizontal X axis units
  --yscale <YSCALE>              Ratio of vertical units to horizontal Y axis units
  --gradient-alg <GRADIENT-ALG>  Algorithm used to compute terrain gradient. GRADIENT-
↪ALG=Horn|ZevenbergenThorne (default: Horn)
  --no-edges                     Do not try to interpolate values at dataset edges or␣
↪close to nodata values
```

Details for options can be found in *gdal raster slope*.

stack

```
* stack [OPTIONS] <INPUTS>...
-----------------------------

Combine together input bands into a multi-band output, either virtual (VRT) or␣
↪materialized.

Positional arguments:
  -i, --input <INPUTS>                                          Input raster datasets␣
↪(or specify a @<filename> to point to a file containing filenames) [1.. values]␣
↪[required]

Options:
  -b, --band <BAND>                                             Input band(s) (1-based␣
↪index) [may be repeated]
  --absolute-path                                               Whether the path to␣
↪the input datasets should be stored as an absolute path
  --resolution <xres>,<yres>|same|average|common|highest|lowest>  Target resolution (in␣
↪destination CRS units) (default: same)
  --bbox <BBOX>                                                 Target bounding box as␣
↪xmin,ymin,xmax,ymax (in destination CRS units)
  --target-aligned-pixels                                       Round target extent to␣
↪target resolution
  --input-nodata <INPUT-NODATA>                                 Set nodata values for␣
↪input bands. [1.. values]
  --output-nodata <OUTPUT-NODATA>                               Set nodata values at␣
↪the destination band level. [1.. values]
  --hide-nodata                                                 Makes the destination␣
↪band not report the NoData.
```

(continues on next page)

(continued from previous page)

```
Advanced Options:
  --if, --input-format <INPUT-FORMAT>                              Input formats [may be␣
↪repeated]
  --oo, --open-option <KEY>=<VALUE>                                Open options [may be␣
↪repeated]
```

Details for options can be found in *gdal raster stack*.

tee

```
* tee [OPTIONS] [<PIPELINE>...]
------------------------------

Pipes the input into the output stream and side nested pipelines.

Positional arguments:
  --tee-pipeline <PIPELINE>  Nested pipeline [1.. values]
```

Details for options can be found in *Output nested pipeline*.

tile

Added in version 3.12.

```
* tile [OPTIONS] <OUTPUT>
-------------------------

Generate tiles in separate files from a raster dataset.

Positional arguments:
  -o, --output <OUTPUT>                                            Output directory␣
↪[required]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>              Output format␣
↪(default: PNG)
  --co, --creation-option <KEY>=<VALUE>                            Creation option [may␣
↪be repeated]
  --tiling-scheme <TILING-SCHEME>                                  Tiling scheme. TILING-
↪SCHEME=raster|WebMercatorQuad|WorldCRS84Quad|WorldMercatorWGS84Quad|GoogleCRS84Quad|PseudoTMS_
↪GlobalMercator|GlobalGeodeticOriginLat270|LINZAntarticaMapTilegrid|APSTILE|CBMTILE|NZTM2000␣
↪(default: WebMercatorQuad)
  --min-zoom <MIN-ZOOM>                                            Minimum zoom level
  --max-zoom <MAX-ZOOM>                                            Maximum zoom level
  --min-x <MIN-X>                                                  Minimum tile X␣
↪coordinate
  --max-x <MAX-X>                                                  Maximum tile X␣
↪coordinate
  --min-y <MIN-Y>                                                  Minimum tile Y␣
↪coordinate
  --max-y <MAX-Y>                                                  Maximum tile Y␣
↪coordinate
```

(continues on next page)

(continued from previous page)

```
  --no-intersection-ok                                          Whether dataset␣
↪extent not intersecting tile matrix is only a warning
  -r, --resampling <RESAMPLING>                                 Resampling method for␣
↪max zoom.␣
↪RESAMPLING=nearest|bilinear|cubic|cubicspline|lanczos|average|rms|mode|min|max|med|q1|q3|sum␣
↪(default: cubic)
  --overview-resampling <OVERVIEW-RESAMPLING>                   Resampling method for␣
↪overviews. OVERVIEW-
↪RESAMPLING=nearest|bilinear|cubic|cubicspline|lanczos|average|rms|mode|min|max|med|q1|q3|sum
  --convention <CONVENTION>                                     Tile numbering␣
↪convention: xyz (from top) or tms (from bottom). CONVENTION=xyz|tms (default: xyz)
  --tile-size <TILE-SIZE>                                       Override default tile␣
↪size
  --add-alpha                                                   Whether to force␣
↪adding an alpha channel
                                                                Mutually exclusive␣
↪with --no-alpha
  --no-alpha                                                    Whether to disable␣
↪adding an alpha channel
                                                                Mutually exclusive␣
↪with --add-alpha
  --output-nodata <OUTPUT-NODATA>                               Output nodata value
  --skip-blank                                                  Do not generate blank␣
↪tiles
  --metadata <KEY>=<VALUE>                                      Add metadata item to␣
↪output tiles [may be repeated]
  --copy-src-metadata                                           Whether to copy␣
↪metadata from source dataset
  --aux-xml                                                     Generate .aux.xml␣
↪sidecar files when needed
  --kml                                                         Generate KML files
  --resume                                                      Generate only missing␣
↪files
  -j, --num-threads <NUM-THREADS>                               Number of jobs (or␣
↪ALL_CPUS) (default: ALL_CPUS)
  --parallel-method <PARALLEL-METHOD>                           Parallelization␣
↪method (thread, spawn, fork). PARALLEL-METHOD=thread|spawn|fork

Advanced Resampling Options:
  --excluded-values <EXCLUDED-VALUES>                           Tuples of values (e.g.
↪ <R>,<G>,<B> or (<R1>,<G1>,<B1>),(<R2>,<G2>,<B2>)) that must beignored as contributing␣
↪source pixels during (average) resampling
  --excluded-values-pct-threshold <EXCLUDED-VALUES-PCT-THRESHOLD>  Minimum percentage of␣
↪source pixels that must be set at one of the --excluded-values to cause the excluded␣
↪value to be used as the target pixel value (default: 50)
  --nodata-values-pct-threshold <NODATA-VALUES-PCT-THRESHOLD>      Minimum percentage of␣
↪source pixels that must be set at one of nodata (or alpha=0 or any other way to␣
↪express transparent pixelto cause the target pixel value to be transparent (default:␣
↪100)

Publication Options:
  --webviewer <WEBVIEWER>                                       Web viewer to␣
```

(continues on next page)

(continued from previous page)

```
↪generate. WEBVIEWER=none|all|leaflet|openlayers|mapml|stac (default: all) [may be␣
↪repeated]
  --url <URL>                                                    URL address where the␣
↪generated tiles are going to be published
  --title <TITLE>                                                Title of the map
  --copyright <COPYRIGHT>                                        Copyright for the map
  --mapml-template <MAPML-TEMPLATE>                              Filename of a␣
↪template mapml file where variables will be substituted
```

Details for options can be found in *gdal raster tile*.

tpi

```
* tpi [OPTIONS]
---------------

Generate a Topographic Position Index (TPI) map

Options:
  -b, --band <BAND>       Input band (1-based index) (default: 1)
  --no-edges              Do not try to interpolate values at dataset edges or close to␣
↪nodata values
```

Details for options can be found in *gdal raster tpi*.

tri

```
* tri [OPTIONS]
---------------

Generate a Terrain Ruggedness Index (TRI) map

Options:
  -b, --band <BAND>        Input band (1-based index) (default: 1)
  --algorithm <ALGORITHM>  Algorithm to compute TRI. ALGORITHM=Riley|Wilson (default:␣
↪Riley)
  --no-edges               Do not try to interpolate values at dataset edges or close to␣
↪nodata values
```

Details for options can be found in *gdal raster tri*.

unscale

```
* unscale [OPTIONS]
-------------------

Convert scaled values of a raster dataset into unscaled values.

Options:
  --ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>  Output data type. OUTPUT-DATA-
↪TYPE=UInt8|Int8|UInt16|Int16|UInt32|Int32|UInt64|Int64|CInt16|CInt32|Float16|Float32|Float64|CFloat32
```

Details for options can be found in *gdal raster unscale*.

update

```
* update [OPTIONS] <OUTPUT>
---------------------------

Update the destination raster with the content of the input one.

Positional arguments:
  -o, --output <OUTPUT>                      Output raster dataset [required]

Options:
  --geometry <GEOMETRY>                      Clipping geometry (WKT or GeoJSON)
  --geometry-crs <GEOMETRY-CRS>              CRS of clipping geometry
  -r, --resampling <RESAMPLING>              Resampling method.␣
↪RESAMPLING=nearest|bilinear|cubic|cubicspline|lanczos|average|rms|mode|min|max|med|q1|q3|sum␣
↪(default: nearest)
  --no-update-overviews                      Do not update existing overviews

Advanced Options:
  --wo, --warp-option <NAME>=<VALUE>         Warping option(s) [may be repeated]
  --to, --transform-option <NAME>=<VALUE>    Transform option(s) [may be repeated]
  --et, --error-threshold <ERROR-THRESHOLD>  Error threshold
```

Details for options can be found in *gdal raster update*.

viewshed

```
* viewshed [OPTIONS]
--------------------

Compute the viewshed of a raster dataset.

Options:
  -p, --pos, --position <X,Y> or <X,Y,H>           Observer position [2..3 values]
  -z, --height <HEIGHT>                            Observer height
  --sd-filename <SD-FILENAME>                      Filename of standard-deviation raster
  --target-height <TARGET-HEIGHT>                  Height of the target above the DEM␣
↪surface in the height unit of the DEM. (default: 0)
  --mode <MODE>                                    Sets what information the output␣
↪contains.. MODE=normal|DEM|ground|cumulative (default: normal)
  --max-distance <MAX-DISTANCE>                    Maximum distance from observer to␣
↪compute visibility. It is also used to clamp the extent of the output raster.
  --min-distance <MIN-DISTANCE>                    Mask all cells less than this␣
↪distance from the observer. Must be less than 'max-distance'.
  --start-angle <START-ANGLE>                      Mask all cells outside of the arc (
↪'start-angle', 'end-angle'). Clockwise degrees from north. Also used to clamp the␣
↪extent of the output raster.
  --end-angle <END-ANGLE>                          Mask all cells outside of the arc (
↪'start-angle', 'end-angle'). Clockwise degrees from north. Also used to clamp the␣
↪extent of the output raster.
  --high-pitch <HIGH-PITCH>                        Mark all cells out-of-range where the␣
```

(continues on next page)

(continued from previous page)

```
↪observable height would be higher than the 'high-pitch' angle from the observer.␣
↪Degrees from horizontal - positive is up. Must be greater than 'low-pitch'.
  --low-pitch <LOW-PITCH>                          Bound observable height to be no␣
↪lower than the 'low-pitch' angle from the observer. Degrees from horizontal - positive␣
↪is up. Must be less than 'high-pitch'.
  --curvature-coefficient <CURVATURE-COEFFICIENT>  Coefficient to consider the effect of␣
↪the curvature and refraction.
  -b, --band <BAND>                                Input band (1-based index) (default:␣
↪1)
  --visible-value <VISIBLE-VALUE>                  Pixel value to set for visible areas␣
↪(default: 255)
  --invisible-value <INVISIBLE-VALUE>              Pixel value to set for invisible␣
↪areas (default: 0)
  --maybe-visible-value <MAYBE-VISIBLE-VALUE>      Pixel value to set for potentially␣
↪visible areas (default: 2)
  --out-of-range-value <OUT-OF-RANGE-VALUE>        Pixel value to set for the cells that␣
↪fall outside of the range specified by the observer location and the maximum distance␣
↪(default: 0)
  --output-nodata <OUTPUT-NODATA>                  The value to be set for the cells in␣
↪the output raster that have no data.
  --observer-spacing <OBSERVER-SPACING>            Cell Spacing between observers␣
↪(default: 10)
  -j, --num-threads <NUM-THREADS>                  Number of jobs (or ALL_CPUS)␣
↪(default: 3)
```

Details for options can be found in *gdal raster viewshed*.

write

```
* write [OPTIONS] <OUTPUT>
--------------------------

Write a raster dataset.

Positional arguments:
  -o, --output <OUTPUT>                                Output raster dataset [required]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)
  --co, --creation-option <KEY>=<VALUE>                Creation option [may be repeated]
  --overwrite                                          Whether overwriting existing␣
↪output dataset is allowed
                                                       Mutually exclusive with --append
  --append                                             Append as a subdataset to␣
↪existing output
                                                       Mutually exclusive with --
↪overwrite
```

Details for options can be found in *gdal raster pipeline write*.

GDALG output (on-the-fly / streamed dataset)

A pipeline can be serialized as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

The `command_line` member of the JSON file should nominally be the whole command line without the final `write` step, and is what is generated by `gdal raster pipeline ! .... ! write out.gdalg.json`.

```
{
    "type": "gdal_streamed_alg",
    "command_line": "gdal raster pipeline ! read in.tif ! reproject --output-
↪crs=EPSG:32632"
}
```

The final `write` step can be added but if so it must explicitly specify the `stream` output format and a non-significant output dataset name.

```
{
    "type": "gdal_streamed_alg",
    "command_line": "gdal raster pipeline ! read in.tif ! reproject --output-
↪crs=EPSG:32632 ! write --output-format=streamed streamed_dataset"
}
```

Substitutions

Added in version 3.12.

It is possible to use **`gdal pipeline`** to use a pipeline already serialized in a .gdal.json file, and customize its existing steps, typically changing an input filename, specifying an output filename, or adding/modifying arguments of steps.

See *Substitutions*.

Nested pipeline

Added in version 3.12.

It is possible to create "nested pipelines", i.e. pipelines inside pipelines.

A nested pipeline is delimited by square brackets (`[` and `]`) surrounded by a space character.

There are 2 kinds of nested pipelines:

- input nested pipelines: where the result dataset of the nested pipeline is used as the input dataset for an argument of the main pipeline.
- output nested pipelines: where the output of a step of the main pipeline is used as the input of the nested pipeline in a following step. Output nested pipelines can only be used with the `tee` step.

See *Nested pipeline*.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Reproject a GeoTIFF file to CRS EPSG:32632 ("WGS 84 / UTM zone 32N") and adding a metadata item

```
$ gdal raster pipeline ! read in.tif ! reproject --output-crs=EPSG:32632 ! edit --
↪metadata AUTHOR=EvenR ! write out.tif --overwrite
```

Example 2: Serialize the command of a reprojection of a GeoTIFF file in a GDALG file, and later read it

```
$ gdal raster pipeline ! read in.tif ! reproject --output-crs=EPSG:32632 ! write in_epsg_
↪32632.gdalg.json --overwrite
$ gdal raster info in_epsg_32632.gdalg.json
```

Example 3: Mosaic on-the-fly several input files and tile that mosaic.

```
gdal raster pipeline ! mosaic input*.tif ! tile output_folder
```

Example 4: Reclassify GeoTIFF and render it as RGB image.

```
$ gdal raster pipeline ! read in.tif ! reclassify -m "[1,10]=1; [11,20]=2; [21,30]=3;␣
↪DEFAULT=NO_DATA" --ot=Byte ! color-map --color-map=color_map.txt --color-
↪selection=exact --add-alpha ! write -f WEBP rendered.webp
```

5.1.3.29 gdal raster pixel-info

Added in version 3.11.

Synopsis

```
Usage: gdal raster pixel-info [OPTIONS] <INPUT> [<column,line> or <X,Y>]

Return information on a pixel of a raster dataset.

Positional arguments:
  -i, --dataset, --input <INPUT>                      Input raster datasets [required]
  -p, --pos, --position <column,line> or <X,Y>        Pixel position [may be repeated]
                                                      Mutually exclusive with --
↪position-dataset

Common Options:
  -h, --help                                          Display help message and exit
  --json-usage                                        Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                              Configuration option [may be␣
↪repeated]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format (default is 'GeoJSON
```

(continues on next page)

(continued from previous page)

```
↪' if 'position-dataset' not specified) [not available in pipelines]
  --position-dataset <POSITION-DATASET>                Vector dataset with coordinates
                                                       Mutually exclusive with --position
  -l, --layer, --input-layer <INPUT-LAYER>             Input layer name
  -o, --output <OUTPUT>                                Output vector dataset (created by␣
↪algorithm)
  --co, --creation-option <KEY>=<VALUE>                Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>         Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                          Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  -b, --band <BAND>                                    Input band(s) (1-based index)␣
↪[may be repeated]
  --overview <OVERVIEW>                                Which overview level of source␣
↪file must be used
  --position-crs <POSITION-CRS>                        CRS of position (default is 'pixel
↪' if 'position-dataset' not specified)
  -r, --resampling <RESAMPLING>                        Resampling algorithm for␣
↪interpolation. RESAMPLING=nearest|bilinear|cubic|cubicspline (default: nearest)
  --promote-pixel-value-to-z                           Whether to set the pixel value as␣
↪Z component of GeoJSON geometry
  --include-field <INCLUDE-FIELD>                      Fields from coordinate dataset to␣
↪include in output (special values: ALL and NONE) (default: ALL) [may be repeated]

Advanced Options:
  --oo, --open-option <KEY>=<VALUE>                    Open options [may be repeated]␣
↪[not available in pipelines]
  --if, --input-format <INPUT-FORMAT>                  Input formats [may be repeated]␣
↪[not available in pipelines]
```

Description

gdal raster pixel-info provide a mechanism to query information about a pixel given its location in one of a variety of coordinate systems.

The following items will be reported (when known):

- Input coordinates
- Input coordinates converted to column, line
- Pixel value per selected band(s), with unscaled value
- For VRT files, which file(s) contribute to the pixel value (only for GeoJSON output)

There are 3 possible ways of providing input location(s):

- from input stream, with a "X Y" pair per line, potentially followed by other text.
- though the *--position* argument.
- through a GDAL vector dataset through the *--position-dataset* argument.

There are 2 possibilities of outputting pixel information:

- either on the output stream as CSV or GeoJSON (default to GeoJSON)

- or as a GDAL vector dataset whose name is specified with the `--output` argument.

Since GDAL 3.13, this algorithm can be part of a *gdal pipeline*. In that case, only reading coordinates from a GDAL vector dataset and outputting results to one is supported.

Added in version 3.13.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

> **Note**
>
> However this algorithm is not natively streaming compatible. Consequently a in-memory temporary dataset will be generated, which may cause significant processing time at opening.

The following options are available:

Program-Specific Options

-b <band>

Selects a band to query. Multiple bands can be listed. By default all bands are queried.

-f, --of, --format, --output-format <FORMAT>

Which vector output format to use.

When `--output` is not specified, `csv` or `geojson` are the only two supported choices, with `geojson` being the default. Otherwise, any GDAL-supported vector format with write capabilities is supported.

--position-dataset <COORDINATE-DATASET>

Added in version 3.13.

GDAL-compatible vector dataset from which input coordinates are read. If the input feature has a point geometry, its location is used to query the raster. For other geometry types, the centroid is used. To read all pixel values for non-point geometries, use *gdal raster zonal-stats*. By default, all attribute fields are copied to the output.

If the `--position-crs` argument is specified, the coordinates of the point are interpreted according to the value of that argument.

Otherwise, if the input vector layer has a CRS, coordinates are assumed to be expressed in it.

Otherwise, coordinates are assumed to be in the CRS raster dataset if it has a CRS.

Otherwise, coordinates are assumed to be expressed in the column,line raster space.

-l, --layer, --input-layer <INPUT-LAYER>

Added in version 3.13.

Input layer name of the dataset specified with `--position-dataset` to use.

-o, --output <OUTPUT>

Added in version 3.13.

Output vector dataset (created by algorithm)

--include-field <FIELD>

Added in version 3.13.

Name of the field(s) from *--position-dataset* to include into the output. By default, all fields are included (which corresponds to special value `ALL`). Special value `NONE` can be used to mean that no field from the position dataset must be included in the output.

--ovr, --overview <index>

Query the (overview_level)th overview (overview_level=0 is the 1st overview), instead of the base band. Note that the x,y location (if the coordinate system is pixel/line) must still be given with respect to the base band.

-p, --pos, --position <column,line> or <X,Y>

Required, if *--position-dataset* is not specified and no content is provided through the input stream.

This can be specified either as an option, a positional value after the dataset name, or when called from **gdal**, as (space separated) values provided on the standard input.

By default, when *--position-crs* is not specified, or set to `pixel`, this is a column, line tuple (possibly with fractional part). If *--position-crs* is set to `dataset`, this is a georeferenced coordinate expressed in the CRS of the dataset. If *--position-crs* is specified to a CRS definition, this is a georeferenced coordinate expressed in this CRS.

X means always longitude or easting, Y means always latitude or northing.

Several x,y tuples may be specified.

--position-crs pixel|dataset|<crs-def>

CRS of position, or one of two following special values:

- `pixel` means that the position is set as column, line (default)
- `dataset` means that the position is a georeferenced coordinate expressed in the CRS of the dataset.

--promote-pixel-value-to-z

Added in version 3.13.

Whether to set the pixel value as Z component of GeoJSON geometry. Only applies if a single band is selected, and for GeoJSON output format.

-r, --resampling nearest|bilinear|cubic|cubicspline

Select a sampling algorithm. The default is `nearest`.

The available methods are:

- `nearest` applies a nearest neighbour.
- `bilinear` applies a bilinear convolution kernel.
- `cubic` applies a cubic convolution kernel.
- `cubicspline` applies a B-Spline convolution kernel.

Standard Options

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Reporting on pixel column=5, line=10 on the file byte.tif

```
$ gdal raster pixel-info byte.tif 5 10
{
  "type":"FeatureCollection",
  "crs":{
    "type":"name",
    "properties":{
      "name":"urn:ogc:def:crs:EPSG::26711"
    }
  },
  "features":[
```

(continues on next page)

(continued from previous page)

```
    {
      "type":"Feature",
      "properties":{
        "input_coordinate":[
          5.0,
          10.0
        ],
        "column":5.0,
        "line":10.0,
        "bands":[
          {
            "band_number":1,
            "raw_value":132,
            "unscaled_value":132.0
          }
        ]
      },
      "geometry":{
        "type":"Point",
        "coordinates":[
          441020.0,
          3750720.0
        ]
      }
    }
  ]
}
```

Example 2: Reporting on point at UTM 11N coordinates easting=441320 and northing=3750720 on the file byte.tif

```
$ gdal raster pixel-info --position-crs=dataset byte.tif 441320 3750720
{
  "type":"FeatureCollection",
  "crs":{
    "type":"name",
    "properties":{
      "name":"urn:ogc:def:crs:EPSG::26711"
    }
  },
  "features":[
    {
      "type":"Feature",
      "properties":{
        "input_coordinate":[
          441320.0,
          3750720.0
        ],
        "column":10.0,
        "line":10.0,
        "bands":[
```

(continues on next page)

(continued from previous page)

```
          {
            "band_number":1,
            "raw_value":115,
            "unscaled_value":115.0
          }
        ]
      },
      "geometry":{
        "type":"Point",
        "coordinates":[
          441320.0,
          3750720.0
        ]
      }
    }
  ]
}
```

Example 3: Reporting on point at WGS84 coordinates longitude=-117.6355 and latitude=33.8970 on the file byte.tif, with CSV output format

```
$ gdal raster pixel-info --of=csv --position-crs=WGS84 byte.tif -117.6355 33.8970
Warning 1: Requested to create spatial CSV layer but GEOMETRY layer creation option not␣
↪set. No geometry will be output.
geom_x,geom_y,extra_content,column,line,band_1_raw_value,band_1_unscaled_value
-117.6355,33.897,,10.020546288988,9.98330473474925,115,115
```

Example 4: Reporting on point at WGS84 coordinates provided on the standard input with longitude, latitude order.

```
$ echo -117.6355 33.8970 | gdal raster pixel-info --of=csv --position-crs=WGS84 byte.tif
Warning 1: Requested to create spatial CSV layer but GEOMETRY layer creation option not␣
↪set. No geometry will be output.
geom_x,geom_y,extra_content,column,line,band_1_raw_value,band_1_unscaled_value
-117.6355,33.897,,10.020546288988,9.98330473474925,115,115
```

Example 5: Reading coordinates to extract from an input GeoPackage file and writing the output to a GeoPackage file

```
gdal raster pixel-info --position-dataset input.gpkg --input byte.tif --output output.
↪gpkg
```

Example 6: Getting pixel values from a on-the-fly resized raster dataset from coordinates in input.gpkg.

```
gdal pipeline read byte.tif ! resize --size 50%,50% -r cubic ! pixel-info input.gpkg !␣
↪write output.gpkg
```

Example 7: Getting pixel values from coordinates in a piped vector dataset, using the _ placeholder dataset name

```
gdal pipeline read input.gml ! swap-xy ! pixel-info --input byte.tif --position-dataset _
↪ ! write output.gpkg
```

5.1.3.30 gdal raster polygonize

Added in version 3.11.

Synopsis

```
Usage: gdal raster polygonize [OPTIONS] <INPUT> <OUTPUT>

Create a polygon feature dataset from a raster band.

Positional arguments:
  -i, --input <INPUT>                                    Input raster datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>           Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                               Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                      Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                               Whether appending to existing␣
↪layer is allowed [not available in pipelines]
  --output-layer <OUTPUT-LAYER>                          Output layer name
  -b, --band <BAND>                                      Input band (1-based index)␣
↪(default: 1)
  --attribute-name <ATTRIBUTE-NAME>                      Name of the field with the pixel␣
↪value (default: DN)
  -c, --connect-diagonal-pixels                          Consider diagonal pixels as␣
↪connected
```

(continues on next page)

(continued from previous page)

```
Advanced Options:
  --if, --input-format <INPUT-FORMAT>                  Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                    Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>      Output open options [may be␣
↪repeated] [not available in pipelines]
```

Description

gdal raster polygonize creates vector polygons for all connected regions of pixels in the raster sharing a common pixel value. Each polygon is created with an attribute indicating the pixel value of that polygon. Pixels that are set to the NoData value, or masked using an external mask band, are not included in processing.

The utility can create the output vector dataset if it does not already exist, otherwise it may append to an existing one.

The utility is based on the :*GDALPolygonize()* function which has additional details on the algorithm.

Since GDAL 3.12, this algorithm can be part of a *gdal pipeline*.

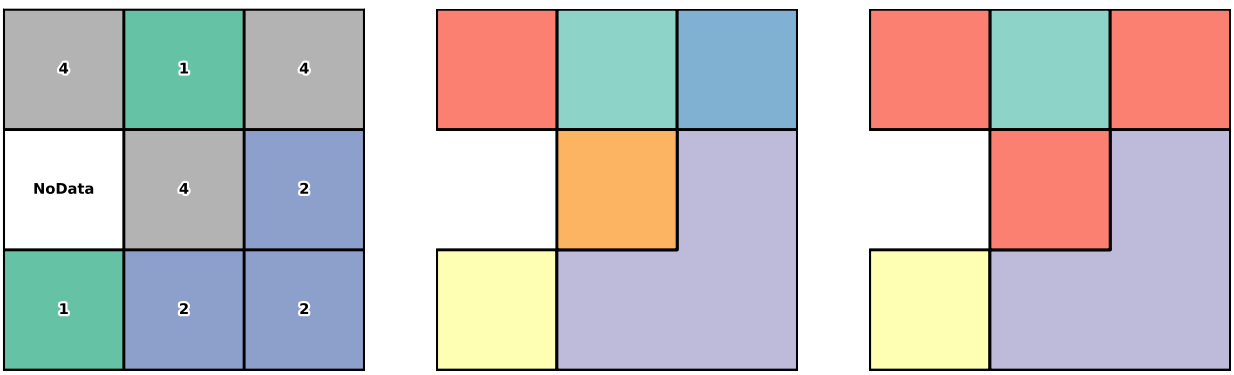


Fig. 2: Polygonization of a 3x3 raster input. The middle figure shows the default behavior, where separate polygons are created for pixel regions that are connected only to a diagonal neighbor. In the figure on the right, `--connect-diagonal-pixels` has been specified, and a single polygon has been created for pixels with value 4.

Added in version 3.12.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

> **Note**
>
> However this algorithm is not natively streaming compatible. Consequently a in-memory temporary dataset will be generated, which may cause significant processing time at opening.

Program-Specific Options

--attribute-name <ELEVATION-NAME>

The name of the field to create (defaults to "DN").

-b, --band <BAND>

Picks a particular band to polygonize. Defaults to band 1.

-c, --connect-diagonal-pixels

Consider diagonal pixels (pixels at the corners) as connected. The default behavior is to only consider pixels that are touching the edges as connected, which is the same as 4-connectivity. When this option is selected, the algorithm will also consider pixels at the corners as connected, which is the same as 8-connectivity.

--nln, --output-layer <OUTPUT-LAYER>

Provides a name for the output vector layer. Defaults to "polygonize".

Standard Options

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--update

Whether to open an existing output dataset in update mode.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Create a shapefile with polygons for the connected regions of band 1 of the input GeoTIFF.

```
gdal raster polygonize input.tif polygonize.shp
```

5.1.3.31 gdal raster proximity

Added in version 3.12.

Synopsis

```
Usage: gdal raster proximity [OPTIONS] <INPUT> <OUTPUT>

Produces a raster proximity map.

Positional arguments:
  -i, --input <INPUT>                                       Input raster datasets␣
↪[required] [not available in pipelines]
  -o, --output <OUTPUT>                                     Output raster dataset␣
↪[required] [not available in pipelines]

Common Options:
  -h, --help                                                Display help message and exit
  --json-usage                                              Display usage as JSON␣
↪document and exit
  --config <KEY>=<VALUE>                                    Configuration option [may be␣
```

(continues on next page)

(continued from previous page)

```
↪repeated]
  -q, --quiet                                              Quiet mode (no progress bar␣
↪or warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>      Output format ("GDALG"␣
↪allowed) [not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                    Creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                              Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                           Mutually exclusive with --
↪append
  --append                                                 Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                           Mutually exclusive with --
↪overwrite
  --ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>  Output data type. OUTPUT-DATA-
↪TYPE=UInt8|Int8|UInt16|Int16|UInt32|Int32|UInt64|Int64|CInt16|CInt32|Float16|Float32|Float64|CFloat32
↪(default: Float32)
  -b, --band <BAND>                                        Input band (1-based index)
  --target-values <TARGET-VALUES>                          Target pixel values [may be␣
↪repeated]
  --distance-units <DISTANCE-UNITS>                        Distance units. DISTANCE-
↪UNITS=pixel|geo (default: pixel)
  --max-distance <MAX-DISTANCE>                            Maximum distance. The nodata␣
↪value will be used for pixels beyond this distance (default: 0)
  --fixed-value <FIXED-VALUE>                              Fixed value for the pixels␣
↪that are within the maximum distance (instead of the actual distance) (default: 0)
  --nodata <NODATA>                                        Specify a nodata value to use␣
↪for pixels that are beyond the maximum distance

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                      Input formats [may be␣
↪repeated] [not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                        Open options [may be␣
↪repeated] [not available in pipelines]
```

Description

gdal raster proximity generates a raster proximity map indicating the Cartesian distance from the center of each pixel to the center of the nearest pixel identified as a target pixel. Target pixels are those in the source raster for which the raster pixel value is in the set of target pixel values.

This subcommand is also available as a potential step of *gdal raster pipeline*

Added in version 3.12.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

> **Note**
>
> However this algorithm is not natively streaming compatible. Consequently a temporary dataset will be generated, which may cause significant processing time at opening.

Program-Specific Options

-b, --band <BAND>

Input band (1-based index)

--distance-units <pixel|geo>

Units for the distance. If *geo* is specified, the distance will be calculated in georeferenced units (meters or degrees) using the pixel size of the input raster. Otherwise, the distance is interpreted in pixels.

--fixed-value <FIXED-VALUE>

Define a fixed value to be written to output pixels that are within *--max-distance* from the target pixels, instead of the actual distance.

--max-distance <MAX-DISTANCE>

Maximum distance to search for a target pixel. The NoData value will be output if no target pixel is found within this distance. Distance is interpreted in pixels unless *--distance-units geo* is specified.

--nodata <NODATA>

Nodata value for the output raster. If not specified, the NoData value of the input band will be used. If the output band does not have a NoData value, then the value 65535 will be used for floating point output types and the maximum value that can be stored will be used for the integer output types.

--target-values <TARGET-VALUES>

A single value or a comma separated list of target pixel values in the source image to be considered target pixels. If not specified, all non-zero pixels will be considered target pixels.

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>

Output data type among `Byte`, `Int8`, `UInt16`, `Int16`, `UInt32`, `Int32`, `UInt64`, `Int64`, `CInt16`, `CInt32`, `Float32`, `Float64`, `CFloat32`, `CFloat64`.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1

title

Proximity map of a raster with max distance of 3 pixels

```
$ gdal raster proximity --max-distance 3  input.tif output.tif
```

Example 2

title

Proximity map of a two bands raster with different target values for each band using a pipeline stack

```
$ gdal raster pipeline stack \
       [ read input.tif ! proximity -b 1 --target-values 1,2,3 ] \
       [ read input.tif ! proximity -b 2 --target-values 4,5,6 ] ! \
       write output.tif
```

5.1.3.32 gdal raster pipeline read

Added in version 3.11.

Description

The `read` operation is for use in a *gdal pipeline* only, and reads a single input raster dataset. This is the first step of a pipeline.

Synopsis

```
* read [OPTIONS] <INPUT>
------------------------

Read a raster dataset.

Positional arguments:
  -i, --input <INPUT>                  Input raster datasets [required]

Advanced Options:
  --if, --input-format <INPUT-FORMAT>  Input formats [may be repeated]
  --oo, --open-option <KEY>=<VALUE>    Open options [may be repeated]
```

Standard Options

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

Examples

Example 1: Read a GeoTIFF file

```
$ gdal raster pipeline read input.tif ! ... [other commands here] ...
```

5.1.3.33 gdal raster reclassify

Added in version 3.11.

Synopsis

```
Usage: gdal raster reclassify [OPTIONS] <INPUT> <OUTPUT>

Reclassify values in a raster dataset

Positional arguments:
  -i, --input <INPUT>                                    Input raster datasets␣
↪[required] [not available in pipelines]
  -o, --output <OUTPUT>                                  Output raster dataset␣
↪[required] [not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
```

(continues on next page)

(continued from previous page)

```
  --json-usage                                              Display usage as JSON␣
↪document and exit
  --config <KEY>=<VALUE>                                    Configuration option [may be␣
↪repeated]
  -q, --quiet                                               Quiet mode (no progress bar␣
↪or warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>       Output format ("GDALG"␣
↪allowed) [not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                     Creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                               Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                            Mutually exclusive with --
↪append
  --append                                                  Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                            Mutually exclusive with --
↪overwrite
  -m, --mapping <MAPPING>                                   Reclassification mappings (or␣
↪specify a @<filename> to point to a file containing mappings [required]
  --ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>  Output data type. OUTPUT-DATA-
↪TYPE=UInt8|Int8|UInt16|Int16|UInt32|Int32|UInt64|Int64|CInt16|CInt32|Float16|Float32|Float64|CFloat32

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                       Input formats [may be␣
↪repeated] [not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                         Open options [may be␣
↪repeated] [not available in pipelines]
```

Description

gdal raster reclassify reclassifies values in an input dataset. A file (or string) specifies the mapping of input pixel values or ranges to output files.

Ranges of values can be specified using brackets:

Notation	Meaning
[a,b]	Closed interval, includes both a and b
[a,b)	Left-closed, right-open, includes a but excludes b
(a,b]	Left-open, right-closed, excludes a but includes b
(a,b)	Open interval, excludes both a and b

An example file is shown below.

```
# remap land cover types
0       = 10      # land
[2,4]   = 20      # freshwater
1       = 40      # ocean
```

(continues on next page)

(continued from previous page)

```
(4,6]    = 50       # forest (excludes 4, includes 5 and 6)
NO_DATA = NO_DATA # leave NoData pixels unmodified
```

The # character indicates a comment that is ignored by the parser but can make the file easier to read.) In this case:

- input values of 0 will be output as 10
- input values between 2 and 4 (inclusive) will be output as 20
- input values of 1 will be output as 40
- input values between 5 and 6 will be output as 50 (4 is excluded)
- NoData values will be preserved as NoData

The presence of any other values in the input will cause an error. If this is not desired, the input range `DEFAULT` can be used to specify an output value for pixels not covered by any other input range. These pixels may be converted unto NoData (`DEFAULT = NO_DATA`), some other constant value (e.g., `DEFAULT = 50`), or left unmodified (`DEFAULT = PASS_THROUGH`).

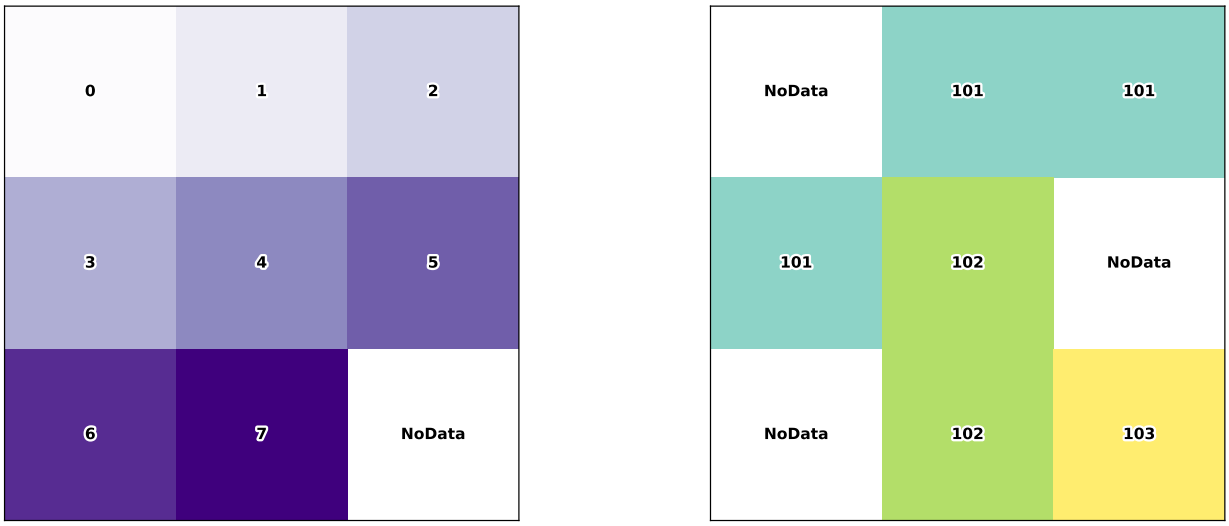


Fig. 3: Raster dataset before (left) and after (right) reclassification with **`gdal raster reclassify --mapping "[1,3]= 101; [4, 5)= 102; 7=102; NO_DATA=103; DEFAULT=NO_DATA"`**.

> **Note**
>
> **`gdal raster reclassify`** supports writing to VRT format; however, VRT files generated in this way can only be opened using GDAL 3.11 or greater.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

-m, --mapping <MAPPING>

A definition of mappings between input and output pixel values, as described above. The mappings may either be provided as text (with each entry separate by a semicolon), or they may be read from a file using `@filename.txt`.

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>

Output data type among `Byte`, `Int8`, `UInt16`, `Int16`, `UInt32`, `Int32`, `UInt64`, `Int64`, `CInt16`, `CInt32`, `Float32`, `Float64`, `CFloat32`, `CFloat64`.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1

```
$ gdal raster reclassify -m "0=10; [2,4]=20; 1=40" -i wbm.tif -o typ.tif
```

5.1.3.34 gdal raster reproject

Added in version 3.11.

Synopsis

```
Usage: gdal raster reproject [OPTIONS] <INPUT> <OUTPUT>

Reproject a raster dataset.

Positional arguments:
  -i, --input <INPUT>                                    Input raster datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output raster dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                         Mutually exclusive with --append
  --append                                               Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                         Mutually exclusive with --
↪overwrite
  -s, --input-crs <INPUT-CRS>                            Input CRS
  --like <DATASET>                                       Dataset to use as a template for␣
↪target bounds, CRS, size and nodata
  -d, --output-crs <OUTPUT-CRS>                          Output CRS
  -r, --resampling <RESAMPLING>                          Resampling method.␣
↪RESAMPLING=nearest|bilinear|cubic|cubicspline|lanczos|average|rms|mode|min|max|med|q1|q3|sum␣
↪(default: nearest)
  --resolution <xres>,<yres>                             Target resolution (in destination␣
↪CRS units)
                                                         Mutually exclusive with --size
```

(continues on next page)

(continued from previous page)

```
  --size <width>,<height>                                Target size in pixels
                                                         Mutually exclusive with --
↪resolution
  --bbox <BBOX>                                          Target bounding box (in␣
↪destination CRS units)
  --bbox-crs <BBOX-CRS>                                  CRS of target bounding box
  -j, --num-threads <NUM-THREADS>                        Number of jobs (or ALL_CPUS)␣
↪(default: ALL_CPUS)

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                    Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                      Open options [may be repeated]␣
↪[not available in pipelines]
  --target-aligned-pixels                                Round target extent to target␣
↪resolution
  --input-nodata <INPUT-NODATA>                          Set nodata values for input bands␣
↪('None' to unset). [1.. values]
  --output-nodata <OUTPUT-NODATA>                        Set nodata values for output␣
↪bands ('None' to unset). [1.. values]
  --add-alpha                                            Adds an alpha mask band to the␣
↪destination when the source raster have none.
  --wo, --warp-option <NAME>=<VALUE>                     Warping option(s) [may be␣
↪repeated]
  --to, --transform-option <NAME>=<VALUE>                Transform option(s) [may be␣
↪repeated]
  --et, --error-threshold <ERROR-THRESHOLD>              Error threshold
```

Description

gdal raster reproject can be used to reproject a raster dataset. The program can reproject to any supported projection.

First, **gdal raster reproject** must determine the extent and resolution of the output, if these have not been specified using *--bbox* and *--resolution*. These are determined by transforming a sample of points from the source CRS to the destination CRS. Details of the procedure can be found in the documentation for *GDALSuggestedWarpOutput()*. If multiple inputs are provided to **gdal raster reproject**, the output extent will be calculated to cover all of them, at a resolution consistent with the highest-resolution input.

Once the dimensions of the output image have been determined, **gdal raster reproject** divides the output image into chunks that can be processed independently. **gdal raster reproject** then iterates over scanlines in these chunks, and for each output pixel determines a rectangular region of source pixels that contribute to the value of the output pixel. The dimensions of this rectangular region are typically determined by estimating the relative scales of the source and destination raster, but can be manually specified (see documentation of the `XSCALE` parameter in *GDALWarpOptions::papszWarpOptions*). Because the source region is a simple rectangle, it is not possible for an output pixel to be associated with source pixels from both sides of the antimeridian or pole (when transforming from geographic coordinates).

The rectangular region of source pixels is then provided to a function that performs the resampling algorithm selected with *--resampling*. Depending on the resampling algorithm and relative scales of the source and destination rasters, source pixels may be weighted either according to the approximate fraction of the source pixel that is covered by the destination pixel (e.g., "mean" and "sum" resampling), or by horizontal and vertical Cartesian distances between the center of the source pixel and the center of the target pixel (e.g., bilinear or cubic spline resampling). In the latter case, the relative weight of an individual source pixel is determined by the product of the weights determined for its row and

column; the diagonal Cartesian distance is not calculated.

This subcommand is also available as a potential step of *gdal raster pipeline*

Program-Specific Options

--add-alpha

Create an output alpha band to identify nodata (unset/transparent) pixels. Value 0 is used for fully transparent pixels. The maximum value for the alpha band, for fully opaque pixels, depends on the data type and the presence of the NBITS band metadata item. If it is present, the maximum value is 2^NBITS-1. Otherwise, if NBITS is not set and the alpha band is of type UInt16 (resp. Int16), 65535 (resp. 32767) is used. Otherwise, 255 is used. The maximum value can also be overridden with `--wo DST_ALPHA_MAX=<value>`.

--bbox `<xmin>,<ymin>,<xmax>,<ymax>`

Set georeferenced extents of output file to be created (in target SRS by default, or in the SRS specified with *--bbox-crs*)

--bbox-crs `<BBOX-CRS>`

Specifies the SRS in which to interpret the coordinates given with *--bbox*.

The coordinate reference systems that can be passed are anything supported by the OGRSpatialReference.SetFromUserInput() call, which includes EPSG Projected, Geographic or Compound CRS (i.e. EPSG:4296), a well known text (WKT) CRS definition, PROJ.4 declarations, or the name of a .prj file containing a WKT CRS definition.

If the SRS has an explicit vertical datum that points to a geoid grid, and the input dataset is a single band dataset, a vertical correction will be applied to the values of the dataset.

This must not be confused with *--output-crs* which is the target SRS of the output dataset. *--bbox-crs* is a convenience e.g. when knowing the output coordinates in a geodetic long/lat SRS, but still wanting a result in a projected coordinate system.

--like `<DATASET>`

Name of GDAL input dataset that serves as a template for default values of options *--size*, *--resolution* *--output-crs* and *--bbox* Note that the pixel values will *not* be copied, and that spatial registration of the template dataset through mechanisms such as GCP, RPC or geolocation array is ignored.

--et, --error-threshold `<ERROR-THRESHOLD>`

Error threshold for transformation approximation, expressed as a number of source pixels. Defaults to 0.125 pixels unless the `RPC_DEM` transformer option is specified, in which case an exact transformer, i.e. `--error-threshold=0`, will be used.

-j, --num-threads `<value>`

Added in version 3.12.

Number of jobs to run at once. Default: number of CPUs detected.

--input-crs, -s `<INPUT-CRS>`

Set input spatial reference. If not specified the SRS found in the input dataset will be used.

The coordinate reference systems that can be passed are anything supported by the OGRSpatialReference.SetFromUserInput() call, which includes EPSG Projected, Geographic or Compound CRS (i.e. EPSG:4296), a well known text (WKT) CRS definition, PROJ.4 declarations, or the name of a .prj file containing a WKT CRS definition.

If the SRS has an explicit vertical datum that points to a geoid grid, and the input dataset is a single band dataset, a vertical correction will be applied to the values of the dataset.

--input-nodata <NODATA>

Set nodata masking values for input bands (different values can be supplied for each band). If more than one value is supplied all values should be quoted to keep them together as a single operating system argument. Masked values will not be used in interpolation (details given in *Nodata / source validity mask handling*)

Use a value of `None` to ignore intrinsic nodata settings on the source dataset.

When this option is set to a non-`None` value, it causes the `UNIFIED_SRC_NODATA` warping option (see *`GDALWarpOptions::papszWarpOptions`*) to be set to `YES`, if it is not explicitly set.

If `--input-nodata` is not explicitly set, but the source dataset has nodata values, they will be taken into account, with `UNIFIED_SRC_NODATA` at `PARTIAL` by default.

--output-crs, -d, <OUTPUT-CRS>

Set output spatial reference. If not specified the SRS found in the input dataset will be used.

The coordinate reference systems that can be passed are anything supported by the OGRSpatialReference.SetFromUserInput() call, which includes EPSG Projected, Geographic or Compound CRS (i.e. EPSG:4296), a well known text (WKT) CRS definition, PROJ.4 declarations, or the name of a .prj file containing a WKT CRS definition.

If the SRS has an explicit vertical datum that points to a geoid grid, and the input dataset is a single band dataset, a vertical correction will be applied to the values of the dataset.

--output-nodata <NODATA>

Set nodata values for output bands (different values can be supplied for each band). If more than one value is supplied all values should be quoted to keep them together as a single operating system argument. New files will be initialized to this value and if possible the nodata value will be recorded in the output file. Use a value of `None` to ensure that nodata is not defined. If this argument is not used then nodata values will be copied from the source dataset. Note that a number of output formats, including GeoTIFF, do not support different per-band nodata values, but a single one for all bands.

-r, --resampling <RESAMPLING>

Resampling method to use. Available methods are:

`nearest`: nearest neighbour resampling (default, fastest algorithm, worst interpolation quality).

`bilinear`: bilinear resampling.

`cubic`: cubic resampling.

`cubicspline`: cubic spline resampling.

`lanczos`: Lanczos windowed sinc resampling.

`average`: average resampling, computes the weighted average of all non-NODATA contributing pixels.

`rms` root mean square / quadratic mean of all non-NODATA contributing pixels

`mode`: mode resampling, selects the value which appears most often of all the sampled points. In the case of ties, the first value identified as the mode will be selected.

`max`: maximum resampling, selects the maximum value from all non-NODATA contributing pixels.

`min`: minimum resampling, selects the minimum value from all non-NODATA contributing pixels.

`med`: median resampling, selects the median value of all non-NODATA contributing pixels.

`q1`: first quartile resampling, selects the first quartile value of all non-NODATA contributing pixels.

`q3`: third quartile resampling, selects the third quartile value of all non-NODATA contributing pixels.

`sum`: compute the weighted sum of all non-NODATA contributing pixels

> **Note**
>
> When downsampling is performed (use of *--resolution* or *--size*), existing overviews (either internal/implicit or external ones) on the source image will be used by default by selecting the closest overview to the desired output resolution. The resampling method used to create those overviews is generally not the one you specify through the *-r* option.

--resolution <xres>,<yres>

Set output file resolution (in target georeferenced units).

If not specified (or not deduced from *--size*), the program will, in the general case, generate an output raster with xres=yres.

If neither *--resolution* nor *--size* are specified, that no reprojection is involved (including taking into account geolocation arrays or RPC), the resolution of the source file(s) will be preserved (in previous version, an output raster with xres=yres was always generated).

Mutually exclusive with *--size*.

--size <width>,<height>

Set output file size in pixels and lines. If width or height is set to 0, the other dimension will be guessed from the computed resolution.

Mutually exclusive with *--resolution*.

--target-aligned-pixels

Align the coordinates of the extent of the output file to the values of the *--resolution*, such that the aligned extent includes the minimum extent (edges lines/columns that are detected as blank, before actual warping, will be removed). Alignment means that xmin / resx, ymin / resy, xmax / resx and ymax / resy are integer values.

--to, --transform-option <NAME>=<VALUE>

Set a transformer option suitable to pass to *GDALCreateGenImgProjTransformer2()*. See *GDALCreateRPCTransformerV2()* for RPC specific options.

To match the gdalwarp -rpc option, use --to METHOD=RPC

--wo, --warp-option <NAME>=<VALUE>

Set a warp option. The *GDALWarpOptions::papszWarpOptions* docs show all options. Multiple options may be listed.

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

-q, --quiet

Suppress progress bar and some warning messages.

Nodata / source validity mask handling

Invalid values in source pixels, either identified through a nodata value metadata set on the source band, a mask band, an alpha band (for an alpha band, a value of 0 means invalid. Other values are used for blending values) or the use of *--input-nodata* will not be used in interpolation. The details of how it is taken into account depends on the resampling kernel:

- for nearest resampling, for each target pixel, the coordinate of its center is projected back to source coordinates and the source pixel containing that coordinate is identified. If this source pixel is invalid, the target pixel is considered as nodata.
- for bilinear, cubic, cubicspline and lanczos, for each target pixel, the coordinate of its center is projected back to source coordinates and a corresponding source pixel is identified. If this source pixel is invalid, the target pixel is considered as nodata (in this case, valid pixels within the kernel radius would not be considered). Given that those resampling kernels have a non-null kernel radius, this source pixel is just one among other several source pixels, and it might be possible that there are invalid values in those other contributing source pixels. The weights used to take into account those invalid values will be set to zero to ignore them.
- for the other resampling methods, source pixels contributing to the target pixel are ignored if invalid. Only the valid ones are taken into account. If there are none, the target pixel is considered as nodata.

Approximate transformation

By default **gdal raster reproject** uses a linear approximator for the transformations with a permitted error of 0.125 pixels in the source dataset. For each processing chunk, the approximator precisely transforms three points per output scanline (the start, middle, and end) from a row and column in the output dataset to a row and column in the source dataset. It then compares a linear approximation of the center point coordinates to the precisely transformed value. If the sum of the horizontal and vertical errors is less than the error threshold then the remaining source points are approximated using linear interpolation between the start and middle point, and between the middle and end point. If the error exceeds the threshold, the scanline is split into two sections and the approximator is recursively applied to each section until the error is less than the threshold or all points have been exactly computed.

Note that a processing chunk does not necessarily correspond to a whole destination scanline. If the output dataset is tiled, its shape will typically be one or several tiles. The `OPTIMIZE_SIZE` warping option can also influence the shape

of the processing chunk. Consequently, for a single (non-zero) value of the error threshold, the linear approximation might result in slightly different values in coordinate transformation, but all of them within the permitted error.

The error threshold (in source dataset pixels) can be controlled with the `--error-threshold` switch. If you want to compare a true pixel-by-pixel reprojection use `--error-threshold=0` which disables this approximator entirely.

Frequently Asked Questions

Q1. Why does the quality of the output looks so bad (no anti-aliasing)?

Did you specify a resampling method, with `--resampling`, other than the default nearest neighbour?

Q2. Why do I get slightly different results whether the output dataset is tiled or not?

This is related to the fact that an approximate coordinate transformation is used by default to speed-up computation. If you want to get the same results whether the output is tiled or not, set `--error-threshold` to zero. Note, however, that this will only work for relatively small images; other factors can still result in different result. See following question (Q3).

Q3. Why do I observe artifacts, that look like resolution changes and are aligned with rectangular areas of the output raster, when warping sufficiently large rasters, particularly in areas where the reprojection involves significant deformation and only with non-nearest resampling ?

The warping engine operates on rectangular areas of the output dataset (generally aligned with tile boundaries for a compressed tile dataset).

During reprojection, a single source pixel does not generally correspond to a single output pixel. The resampling method must therefore properly account for this and compute a ratio between the number of source and target pixels in the horizontal (X) and vertical (Y) directions. These ratios are computed per warping chunk. This maximizes the local quality of the warping but has the downside of creating visual discontinuities between warping chunks.

If you favor a seamless result, you may manually specify the XSCALE and YSCALE warping options with `--wo`. The XSCALE (resp. YSCALE) value is the ratio expressing the resampling factor, i.e. the number of destination pixels per source pixel, along the horizontal (resp. vertical) axis. It equals to one for no resampling, is below one for downsampling, and above one for upsampling.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Reproject a GeoTIFF file to CRS EPSG:32632 ("WGS 84 / UTM zone 32N")

```
$ gdal raster reproject --output-crs=EPSG:32632 in.tif out.tif --overwrite
```

Example 2: Converting a raster that uses an embedded CRS without a known identifier

In this example, the input file has an embedded CRS defined in WKT2 but without an associated identifier, and it will be reprojected to the known ESRI:54052 CRS identifier.

Using "PROFILE=BASELINE" creates a PAM file that stores the CRS using the ESRI:54052 identifier, instead of embedding CRS information in GeoTIFF metadata tags. See *GTiff* PROFILE.

```
$ gdal raster reproject --creation-option "PROFILE=BASELINE" --output-crs=ESRI:54052␣
↪input.tif output.tif --overwrite
```

5.1.3.35 gdal raster resize

Added in version 3.11.

Synopsis

```
Usage: gdal raster resize [OPTIONS] <INPUT> <OUTPUT>

Resize a raster dataset without changing the georeferenced extents.

Positional arguments:
  -i, --input <INPUT>                                    Input raster datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output raster dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                         Mutually exclusive with --append
  --append                                               Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                         Mutually exclusive with --
↪overwrite
  --resolution <xres>,<yres>                             Target resolution (in destination␣
↪CRS units)
                                                         Mutually exclusive with --size
  --size <width[%]>,<height[%]>                          Target size in pixels (or␣
↪percentage if using '%' suffix) [required]
                                                         Mutually exclusive with --
```

(continues on next page)

(continued from previous page)

```
↪resolution
  -r, --resampling <RESAMPLING>                        Resampling method.␣
↪RESAMPLING=nearest|bilinear|cubic|cubicspline|lanczos|average|mode (default: nearest)

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                  Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                    Open options [may be repeated]␣
↪[not available in pipelines]
```

Description

gdal raster resize can be used to resize a raster dataset without changing the georeferenced extents.

This subcommand is also available as a potential step of *gdal raster pipeline*

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the GDALG output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

-r, --resampling <RESAMPLING>

Resampling method to use. Available methods are:

nearest: nearest neighbour resampling (default, fastest algorithm, worst interpolation quality).

bilinear: bilinear resampling.

cubic: cubic resampling.

cubicspline: cubic spline resampling.

lanczos: Lanczos windowed sinc resampling.

average: average resampling, computes the weighted average of all non-NODATA contributing pixels.

--resolution <xres>,<yres>

Added in version 3.12.

Set output file resolution (in target georeferenced units).

Mutually exclusive with *--size*.

--size <width[%]>,<height[%]>

Set output raster width and height, expressed in pixels, or percentage if using the % suffix. If the width or the height is set to 0, the other dimension will be guessed from the computed resolution.

Mutually exclusive with *--resolution*.

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Resize a dataset to 1000 columns and 500 lines using cubic resampling

```
$ gdal raster resize --size=1000,500 -r cubic in.tif out.tif --overwrite
```

Example 2: Resize a dataset to half size using cubic resampling

```
$ gdal raster resize --size=50%,50% -r cubic in.tif out.tif --overwrite
```

5.1.3.36 gdal raster rgb-to-palette

Added in version 3.12.

Synopsis

```
Usage: gdal raster rgb-to-palette [OPTIONS] <INPUT> <OUTPUT>

Convert a RGB image into a pseudo-color / paletted image.

Positional arguments:
  -i, --input <INPUT>                                    Input raster datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output raster dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                         Mutually exclusive with --append
  --append                                               Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                         Mutually exclusive with --
↪overwrite
  --color-count <COLOR-COUNT>                            Select the number of colors in␣
↪the generated color table (default: 256)
  --color-map <COLOR-MAP>                                Color map filename
  --output-nodata <OUTPUT-NODATA>                        Output nodata value
  --no-dither                                            Disable Floyd-Steinberg dithering
  --bit-depth <BIT-DEPTH>                                Bit depth of color palette␣
↪component (8 bit causes longer computation time). BIT-DEPTH=5|8 (default: 5)

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                    Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                      Open options [may be repeated]␣
↪[not available in pipelines]
```

Description

gdal raster rgb-to-palette computes an optimal pseudo-color table for a given RGB image using a median cut algorithm on a downsampled RGB histogram, unless a color table is provided with the *--color-map* option. Then it converts the image into a pseudo-colored image using the color table. This conversion utilizes by default Floyd-Steinberg dithering (error diffusion) to maximize output image visual quality.

This subcommand is also available as a potential step of *gdal raster pipeline*

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

> **Note**
>
> However this algorithm is not natively streaming compatible. Consequently a temporary dataset will be generated, which may cause significant processing time at opening.

Program-Specific Options

--color-count <COLOR-COUNT>

Select the number of colors in the generated color table. Defaults to 256. Must be between 2 and 256.

--color-map <FILENAME>

Extract the color table from <FILENAME> instead of computing it. Can be used to have a consistent color table for multiple files. The <FILENAME> must be either a raster file in a GDAL supported format with a palette or a color file in a supported format (.txt, QGIS .qml, QGIS .qlr).

--output-nodata <VALUE>

Added in version 3.13.

Specifies the output nodata value, that will be used for input pixels at the source nodata value or that are transparent due to a mask band, such as an alpha band. The destination nodata value must be between 0 and 255. If there is a source nodata value or a dataset-wide mask band is attached to the source bands, and *--output-nodata* is not specified, it defaults to 0.

--no-dither

Added in version 3.13.

Disable Floyd-Steinberg dithering (error diffusion). This may be useful to improve the compression rate by avoiding the "noise" added by error diffusion.

--bit-depth 5|8

Added in version 3.13.

Bit depth of each of the output color table R, G, B component. The default value is 5, which means that each of the component is limited to 32 = 2 ^ 5 values. Setting bit depth to 8 increases the fidelity of the output color table at the expense of longer computation time.

Standard Options

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation

for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1

Convert a TIFF file into a paletted PNG image

```
$ gdal raster rgb-to-palette input.tif output.png
```

5.1.3.37 gdal raster roughness

Added in version 3.11.

Synopsis

```
Usage: gdal raster roughness [OPTIONS] <INPUT> <OUTPUT>

Generate a roughness map

Positional arguments:
  -i, --input <INPUT>                                    Input raster datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output raster dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
```

(continues on next page)

(continued from previous page)

```
  --overwrite                                           Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                        Mutually exclusive with --append
  --append                                              Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                        Mutually exclusive with --
↪overwrite
  -b, --band <BAND>                                     Input band (1-based index)␣
↪(default: 1)
  --no-edges                                            Do not try to interpolate values␣
↪at dataset edges or close to nodata values

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                   Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                     Open options [may be repeated]␣
↪[not available in pipelines]
```

Description

gdal raster roughness generates a single-band raster with values computed from the elevation. Roughness is the largest inter-cell difference of a central pixel and its surrounding cell, as defined in Wilson et al. (2007, Marine Geodesy 30:3-35).

This subcommand is also available as a potential step of *gdal raster pipeline*

The roughness value -9999 is used as the nodata value.

A nodata value in the target dataset will also be emitted if at least one pixel set to the nodata value is found in the 3x3 window centered around each source pixel. By default, the algorithm will compute values at image edges or if a nodata value is found in the 3x3 window, by interpolating missing values, unless `--no-edges` is specified, in which case a 1-pixel border around the image will be set with the nodata value.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

-b, --band <BAND>

Index (starting at 1) of the band to which the roughness must be computed.

--no-edges

Do not try to interpolate values at dataset edges or close to nodata values

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Generates a roughness map from a DTED0 file.

```
$ gdal raster roughness n43.dt0 out.tif --overwrite
```

5.1.3.38 gdal raster scale

Added in version 3.11.

Synopsis

```
Usage: gdal raster scale [OPTIONS] <INPUT> <OUTPUT>

Scale the values of the bands of a raster dataset.

Positional arguments:
```

(continues on next page)

(continued from previous page)

```
  -i, --input <INPUT>                                       Input raster datasets␣
↪[required] [not available in pipelines]
  -o, --output <OUTPUT>                                     Output raster dataset␣
↪[required] [not available in pipelines]

Common Options:
  -h, --help                                                Display help message and exit
  --json-usage                                              Display usage as JSON␣
↪document and exit
  --config <KEY>=<VALUE>                                    Configuration option [may be␣
↪repeated]
  -q, --quiet                                               Quiet mode (no progress bar␣
↪or warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>       Output format ("GDALG"␣
↪allowed) [not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                     Creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                               Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                            Mutually exclusive with --
↪append
  --append                                                  Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                            Mutually exclusive with --
↪overwrite
  --ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>  Output data type. OUTPUT-DATA-
↪TYPE=UInt8|Int8|UInt16|Int16|UInt32|Int32|UInt64|Int64|CInt16|CInt32|Float16|Float32|Float64|CFloat32
  -b, --band <BAND>                                         Select band to restrict the␣
↪scaling (1-based index)
  --input-min <INPUT-MIN>                                   Minimum value of the source␣
↪range
                                                            Depends on --input-max
  --input-max <INPUT-MAX>                                   Maximum value of the source␣
↪range
                                                            Depends on --input-min
  --output-min <OUTPUT-MIN>                                 Minimum value of the␣
↪destination range
                                                            Depends on --output-max
  --output-max <OUTPUT-MAX>                                 Maximum value of the␣
↪destination range
                                                            Depends on --output-min
  --exponent <EXPONENT>                                     Exponent to apply non-linear␣
↪scaling with a power function
  --no-clip                                                 Do not clip input values to␣
↪[innput-min, input-max]

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                       Input formats [may be␣
↪repeated] [not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                         Open options [may be␣
```

(continues on next page)

(continued from previous page)

```
↪repeated] [not available in pipelines]
```

Description

gdal raster scale can be used to rescale the input pixels values from the range *--input-min* to *--input-max* to the range *--output-min* to *--output-max*. It is also often necessary to reset the output datatype with the *--ot* switch. If omitted the output range is from the minimum value to the maximum value allowed for integer data types (for example from 0 to 255 for Byte output) or from 0 to 1 for floating-point data types. If omitted the input range is automatically computed from the source dataset. This may be a slow operation on a large source dataset, and if using it multiple times for several gdal_translate invocation, it might be beneficial to call `gdal raster edit --stats {source_dataset}` priorly to precompute statistics, for formats that support serializing statistics computations (GeoTIFF, VRT...) Source values are clipped to the range defined by `srcmin` and `srcmax`, unless *--no-clip* is set.

By default, the scaling is applied to all bands. It is possible to restrict it to a single band with *--band* and leave values of other bands unmodified.

This command is the reverse operation of *gdal raster unscale*.

This subcommand is also available as a potential step of *gdal raster pipeline*

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

-b, --band <BAND>

Index (starting at 1) of the band to which the scaling must be only applied.

--exponent <EXPONENT>

Apply non-linear scaling with a power function. `exp_val` is the exponent of the power function (must be positive). This option must be used with the *--input-min* / *--input-max* / *--output-min* / *--output-max* options.

The scaled value `Dst` is calculated from the source value `Src` with the following formula:

$$Dst = (Dst_{max} - Dst_{min}) \times \max\left(0, \min\left(1, \left(\frac{Src - Src_{min}}{Src_{max} - Src_{min}}\right)^{exp_val}\right)\right) + Dst_{min}$$

> **Note**
>
> *gdal raster unscale* assumes linear scaling, and this cannot unscale values back to the original ones.

--input-max <MAX>

Maximum value of the source range. If not specified, it will be calculated from the source dataset. This option must be used together with *--input-min*.

--input-min <MIN>

Minimum value of the source range. If not specified, it will be calculated from the input dataset. This option must be used together with *--input-max*.

--output-max <MAX>

Maximum value of the output range. This option must be used together with *--output-min*.

--output-min <MIN>

Minimum value of the output range. This option must be used together with *--output-max*.

--no-clip

Disable clipping input values to the source range. Note that using this option with non-linear scaling with a non-integer exponent will cause input values lower than the minimum value of the source range to be mapped to not-a-number.

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>

Output data type among `Byte`, `Int8`, `UInt16`, `Int16`, `UInt32`, `Int32`, `UInt64`, `Int64`, `CInt16`, `CInt32`, `Float32`, `Float64`, `CFloat32`, `CFloat64`.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Rescale linearly values of a UInt16 dataset from [0,4095] to a Byte dataset [0,255]

```
$ gdal raster scale --datatype Byte --input-min 0 --input-max 4095 uint16.tif byte.tif --
↪overwrite
```

5.1.3.39 gdal raster select

Added in version 3.11.

Synopsis

```
Usage: gdal raster select [OPTIONS] <INPUT> <OUTPUT> <BAND>

Select a subset of bands from a raster dataset.

Positional arguments:
  -i, --input <INPUT>                                  Input raster datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                Output raster dataset [required]␣
↪[not available in pipelines]
  -b, --band <BAND>                                    Band(s) (1-based index, 'mask',
↪'mask:<band>' or color interpretation such as 'red') [1.. values] [required]

Common Options:
  -h, --help                                           Display help message and exit
  --json-usage                                         Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                               Configuration option [may be␣
↪repeated]
  -q, --quiet                                          Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                Creation option [may be repeated]␣
↪[not available in pipelines]
  --overwrite                                          Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                       Mutually exclusive with --append
  --append                                             Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                       Mutually exclusive with --
↪overwrite
  --exclude                                            Exclude specified bands
  --mask <MASK>                                        Mask band (1-based index, 'mask',
```

(continues on next page)

(continued from previous page)

```
↪'mask:<band>' or 'none')

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                  Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                    Open options [may be repeated]␣
↪[not available in pipelines]
```

Description

gdal raster select can be used to select and re-order a subset of raster bands from a raster dataset.

This subcommand is also available as a potential step of *gdal raster pipeline*

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the GDALG output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

--exclude

Added in version 3.13.

Modifies the behavior of the algorithm such that all bands are selected, except the ones mentioned by *--band*.

--band <BAND>

Select one or several input bands. Bands are numbered from 1. Multiple values may be selected to select a set of input bands to write to the output file, or to reorder bands. The special value mask means the mask band of the first band of the input dataset. The mask band of any band can be specified with mask:<band>. A mask band selected with --band will become an ordinary band in the output dataset.

Starting with GDAL 3.13, <BAND> can also be a color interpretation such as "red", "green", "blue", "alpha", "nir", etc.

--mask <BAND>

Select one input band to create output dataset mask band.. Bands are numbered from 1. The special value mask means the mask band of the first band of the input dataset. The mask band of any band can be specified with mask:<band>. <BAND> can be set to none to avoid copying the global mask of the input dataset if it exists. Otherwise it is copied by default, unless the mask is an alpha channel, or if it is explicitly selected to be a regular band of the output dataset (--band mask)

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Reorder a 3-band dataset with bands ordered Blue, Green, Red to Red, Green, Blue

```
$ gdal raster select --band 3,2,1 bgr.tif rgb.tif --overwrite
```

Example 2: Select input bands by their color interpretation

```
$ gdal raster select --band red,green,blue input.tif rgb.tif --overwrite
```

Example 3: Convert a RGBA dataset to a YCbCR JPEG compressed GeoTIFF

```
$ gdal raster select --band 1,2,3 --mask 4 --co COMPRESS=JPEG,PHOTOMETRIC=YCBCR rgba.tif␣
↪rgb_mask.tif
```

5.1.3.40 gdal raster set-type

Added in version 3.11.

Synopsis

```
Usage: gdal raster set-type [OPTIONS] <INPUT> <OUTPUT>

Modify the data type of bands of a raster dataset.

Positional arguments:
  -i, --input <INPUT>                                       Input raster datasets␣
↪[required] [not available in pipelines]
  -o, --output <OUTPUT>                                     Output raster dataset␣
↪[required] [not available in pipelines]

Common Options:
  -h, --help                                                Display help message and exit
  --json-usage                                              Display usage as JSON␣
↪document and exit
  --config <KEY>=<VALUE>                                    Configuration option [may be␣
↪repeated]
  -q, --quiet                                               Quiet mode (no progress bar␣
↪or warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>       Output format ("GDALG"␣
↪allowed) [not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                     Creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                               Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                            Mutually exclusive with --
↪append
  --append                                                  Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                            Mutually exclusive with --
↪overwrite
  --ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>   Output data type. OUTPUT-DATA-
↪TYPE=UInt8|Int8|UInt16|Int16|UInt32|Int32|UInt64|Int64|CInt16|CInt32|Float16|Float32|Float64|CFloat32
↪[required]

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                       Input formats [may be␣
↪repeated] [not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                         Open options [may be␣
↪repeated] [not available in pipelines]
```

Description

gdal raster set-type can be used to force the output image bands to have a specific data type. Values may be truncated or rounded if the output data type is "smaller" than the input one.

This subcommand is also available as a potential step of *gdal raster pipeline*

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

--ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>

Output data type among `Byte`, `Int8`, `UInt16`, `Int16`, `UInt32`, `Int32`, `UInt64`, `Int64`, `CInt16`, `CInt32`, `Float32`, `Float64`, `CFloat32`, `CFloat64`.

Standard Options

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Convert to Float32 data type

```
$ gdal raster set-type --datatype Float32 byte.tif float32.tif --overwrite
```

5.1.3.41 gdal raster slope

Added in version 3.11.

Synopsis

```
Usage: gdal raster slope [OPTIONS] <INPUT> <OUTPUT>

Generate a slope map
```

(continues on next page)

(continued from previous page)

```
Positional arguments:
  -i, --input <INPUT>                                   Input raster datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                 Output raster dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                            Display help message and exit
  --json-usage                                          Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                Configuration option [may be␣
↪repeated]
  -q, --quiet                                           Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                 Creation option [may be repeated]␣
↪[not available in pipelines]
  --overwrite                                           Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                        Mutually exclusive with --append
  --append                                              Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                        Mutually exclusive with --
↪overwrite
  -b, --band <BAND>                                     Input band (1-based index)␣
↪(default: 1)
  --unit <UNIT>                                         Unit in which to express slopes.␣
↪UNIT=degree|percent (default: degree)
  --xscale <XSCALE>                                     Ratio of vertical units to␣
↪horizontal X axis units
  --yscale <YSCALE>                                     Ratio of vertical units to␣
↪horizontal Y axis units
  --gradient-alg <GRADIENT-ALG>                         Algorithm used to compute terrain␣
↪gradient. GRADIENT-ALG=Horn|ZevenbergenThorne (default: Horn)
  --no-edges                                            Do not try to interpolate values␣
↪at dataset edges or close to nodata values

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                   Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                     Open options [may be repeated]␣
↪[not available in pipelines]
```

Description

gdal raster slope generates a slope map, from any GDAL-supported elevation raster.

This subcommand is also available as a potential step of *gdal raster pipeline*

It generates a 32-bit float raster with slope values. You have the option of specifying the type of slope value you want: degrees or percent slope. In cases where the horizontal units differ from the vertical units, you can also supply scaling factors to account for differences between vertical and horizontal units.

The value *-9999* is used as the output nodata value. A nodata value in the target dataset will be emitted if at least one pixel set to the nodata value is found in the 3x3 window centered around each source pixel. By default, the algorithm will compute values at image edges or if a nodata value is found in the 3x3 window, by interpolating missing values, unless `--no-edges` is specified, in which case a 1-pixel border around the image will be set with the nodata value.

In general, it assumes that x, y and z units are identical. However, if none of `--xscale` and `--yscale` are specified, and the CRS is a geographic or projected CRS, it will automatically determine the appropriate ratio from the units of the CRS, as well as the potential value of the units of the raster band (as returned by `GDALRasterBand::GetUnitType()`, if it is metre, foot international or US survey foot). Note that for geographic CRS, the result for source datasets at high latitudes may be incorrect, and prior reprojection to a polar projection might be needed using *gdal raster reproject*.

If x (east-west) and y (north-south) units are identical, but z (elevation) units are different, the `--xscale` and `--yscale` can be used to set the ratio of vertical units to horizontal. For geographic CRS near the equator, where units of latitude and units of longitude are similar, elevation (z) units can be converted to be compatible by using scale=370400 (if elevation is in feet) or scale=111120 (if elevation is in meters). For locations not near the equator, the `--xscale` value can be taken as the `--yscale` value multiplied by the cosine of the mean latitude of the raster.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

-b, --band <BAND>

Index (starting at 1) of the band to which the slope must be computed.

--gradient-alg `Horn|ZevenbergenThorne`

Algorithm used to compute terrain gradient. The default is `Horn`. The literature suggests Zevenbergen & Thorne to be more suited to smooth landscapes, whereas Horn's formula to perform better on rougher terrain.

--no-edges

Do not try to interpolate values at dataset edges or close to nodata values

--unit `degree|percent`

Unit in which to express slopes. Defaults to `degree`.

--xscale <scale>

Ratio of vertical units to horizontal X axis units. If the horizontal unit of the source DEM is degrees (e.g Lat/Long WGS84 projection), you can use scale=111120 if the vertical units are meters (or scale=370400 if they are in feet).

If none of `--xscale` and `--yscale` are specified, and the CRS is a geographic or projected CRS, **gdal raster slope** will automatically determine the appropriate ratio from the units of the CRS, as well as the potential value of the units of the raster band (as returned by `GDALRasterBand::GetUnitType()`, if it is metre, foot international or US survey foot). Note that for geographic CRS, the result for source datasets at high latitudes may be incorrect, and prior reprojection to a polar projection might be needed.

If `--xscale` is specified, `--yscale` must also be specified.

--yscale <scale>

Ratio of vertical units to horizontal Y axis units. If the horizontal unit of the source DEM is degrees (e.g Lat/Long

WGS84 projection), you can use scale=111120 if the vertical units are meters (or scale=370400 if they are in feet)

If none of `--xscale` and `--yscale` are specified, and the CRS is a geographic or projected CRS, **gdal raster slope** will automatically determine the appropriate ratio from the units of the CRS, as well as the potential value of the units of the raster band (as returned by *GDALRasterBand::GetUnitType()*, if it is metre, foot international or US survey foot). Note that for geographic CRS, the result for source datasets at high latitudes may be incorrect, and prior reprojection to a polar projection might be needed.

If `--yscale` is specified, `--xscale` must also be specified.

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Generates a slope map from a DTED0 file.

```
$ gdal raster slope n43.dt0 out.tif --overwrite
```

5.1.3.42 gdal raster sieve

Added in version 3.11.

Synopsis

```
Usage: gdal raster sieve [OPTIONS] <INPUT> <OUTPUT>

Remove small polygons from a raster dataset.

Positional arguments:
  -i, --input <INPUT>                                    Input raster datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output raster dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                         Mutually exclusive with --append
  --append                                               Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                         Mutually exclusive with --
↪overwrite
  --mask <MASK>                                          Use the first band of the␣
↪specified file as a validity mask (all pixels with a value other than zero will be␣
↪considered suitable for inclusion in polygons)
  -b, --band <BAND>                                      Input band (1-based index)
  -s, --size-threshold <SIZE-THRESHOLD>                  Minimum size of polygons to keep␣
↪(default: 2)
  -c, --connect-diagonal-pixels                          Consider diagonal pixels as␣
↪connected

Advanced Options:
```

(continues on next page)

(continued from previous page)

```
  --if, --input-format <INPUT-FORMAT>                    Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                      Open options [may be repeated]␣
↪[not available in pipelines]
```

Description

gdal raster sieve removes raster polygons smaller than a provided threshold size (in pixels) and replaces them with the pixel value of the largest neighbour polygon.

The input dataset is read as integer data which means that floating point values are rounded to integers. Re-scaling source data may be necessary in some cases (e.g. 32-bit floating point data with min=0 and max=1).

This subcommand is also available as a potential step of *gdal raster pipeline* (since GDAL 3.12)

Added in version 3.12.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

> **Note**
>
> However this algorithm is not natively streaming compatible. Consequently a temporary dataset will be generated, which may cause significant processing time at opening.

Program-Specific Options

-b, --band `<BAND>`

Input band (1-based index)

-c, --connect-diagonal-pixels

Consider diagonal pixels (pixels at the corners) as connected. The default behavior is to only consider pixels that are touching the edges as connected, which is the same as 4-connectivity. When this option is selected, the algorithm will also consider pixels at the corners as connected, which is the same as 8-connectivity.

--mask `<MASK>`

Use the first band of the specified file as a validity mask: all pixels in the mask band with a value other than zero will be considered suitable for inclusion in polygons.

-s, --size-threshold `<SIZE-THRESHOLD>`

Minimum size of polygons to keep (default: 2)

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Remove polygons smaller than 10 pixels from the band 2 of a raster.

```
$ gdal raster sieve -b 2 -s 10 input.tif output.tif
```

5.1.3.43 gdal raster stack

Added in version 3.11.

Synopsis

```
Usage: gdal raster stack [OPTIONS] <INPUTS>... <OUTPUT>

Combine together input bands into a multi-band output, either virtual (VRT) or␣
↪materialized.
```

(continues on next page)

(continued from previous page)

```
Positional arguments:
  -i, --input <INPUTS>                                          Input raster datasets␣
↪(or specify a @<filename> to point to a file containing filenames) [1.. values]␣
↪[required] [not available in pipelines]
  -o, --output <OUTPUT>                                         Output raster dataset␣
↪[required] [not available in pipelines]

Common Options:
  -h, --help                                                    Display help message␣
↪and exit
  --json-usage                                                  Display usage as JSON␣
↪document and exit
  --config <KEY>=<VALUE>                                        Configuration option␣
↪[may be repeated]
  -q, --quiet                                                   Quiet mode (no␣
↪progress bar or warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>           Output format ("GDALG"␣
↪allowed) [not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                         Creation option [may␣
↪be repeated] [not available in pipelines]
  --overwrite                                                   Whether overwriting␣
↪existing output dataset is allowed [not available in pipelines]
                                                                Mutually exclusive␣
↪with --append
  --append                                                      Append as a subdataset␣
↪to existing output [not available in pipelines]
                                                                Mutually exclusive␣
↪with --overwrite
  -b, --band <BAND>                                             Input band(s) (1-based␣
↪index) [may be repeated]
  --absolute-path                                               Whether the path to␣
↪the input datasets should be stored as an absolute path
  --resolution <xres>,<yres>|same|average|common|highest|lowest>  Target resolution (in␣
↪destination CRS units) (default: same)
  --bbox <BBOX>                                                 Target bounding box as␣
↪xmin,ymin,xmax,ymax (in destination CRS units)
  --target-aligned-pixels                                       Round target extent to␣
↪target resolution
  --input-nodata <INPUT-NODATA>                                 Set nodata values for␣
↪input bands. [1.. values]
  --output-nodata <OUTPUT-NODATA>                               Set nodata values at␣
↪the destination band level. [1.. values]
  --hide-nodata                                                 Makes the destination␣
↪band not report the NoData.

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                           Input formats [may be␣
↪repeated] [not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                             Open options [may be␣
↪repeated] [not available in pipelines]
```

Description

This program combine together input bands from GDAL datasets into a dataset, that can be either a virtual stack in the *VRT (Virtual Dataset)* format, or in a more conventional raster format such as GeoTIFF.

All bands of each input file are added as separate output bands, unless `-b` is specified to select a subset of them.

The list of input GDAL datasets can be specified at the end of the command line or put in a text file (one filename per line) for very long lists. Wildcards '*', '?' or '[' of `VSIGlob()` can be used, even on files located on network file systems such as /vsis3/, /vsigs/, /vsiaz/, etc.

Stating with GDAL 3.12, this command can also be used as the first step of *gdal raster pipeline*.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

--absolute-path

Added in version 3.12.0.

When writing a VRT file, enables writing the absolute path of the input datasets. By default, input filenames are written in a relative way with respect to the VRT filename (when possible).

-b, --band <band>

Select an input <band> to be processed. Bands are numbered from 1. If input bands not set all bands will be added to the output. Multiple `-b` switches may be used to select a set of input bands.

--bbox <xmin>,<ymin>,<xmax>,<ymax>

Set georeferenced extents of output file. The values must be expressed in georeferenced units. If not specified, the extent of the output is the minimum bounding box of the set of source rasters. Pixels within the extent of the output but not covered by a source raster will be read as valid pixels with a value of zero unless a NODATA value is specified using `--output-nodata`.

--hide-nodata

Even if any band contains nodata value, giving this option makes the output band not report the NoData. Useful when you want to control the background color of the dataset.

--input-nodata <value>[,<value>]...

Set nodata values for input bands (different values can be supplied for each band). If the option is not specified, the intrinsic nodata settings on the source datasets will be used (if they exist). The value set by this option is written in the NODATA element of each `ComplexSource` element.

--output-nodata <value>[,<value>]...

Set nodata values at the output band level (different values can be supplied for each band). If more than one value is supplied, all values should be quoted to keep them together as a single operating system argument. If the option is not specified, intrinsic nodata settings on the first dataset will be used (if they exist). The value set by this option is written in the `NoDataValue` element of each `VRTRasterBand element`. Use a value of *None* to ignore intrinsic nodata settings on the source datasets.

--resolution {<xres,yres>|same|highest|lowest|average}

In case the resolution of all input files is not the same, the `--resolution` flag enables the user to control the way the output resolution is computed.

same, the default, checks that all source rasters have the same resolution and errors out when this is not the case.

highest will pick the smallest values of pixel dimensions within the set of source rasters.

lowest will pick the largest values of pixel dimensions within the set of source rasters.

average will compute an average of pixel dimensions within the set of source rasters.

common determines the greatest common divisor of the source pixel dimensions, e.g. 0.2 for source pixel dimensions of 0.4 and 0.6.

<xres>,<yres>. The values must be expressed in georeferenced units. Both must be positive values.

--target-aligned-pixels

(target aligned pixels) align the coordinates of the extent of the output file to the values of the `--resolution`, such that the aligned extent includes the minimum extent. Alignment means that xmin / resx, ymin / resy, xmax / resx and ymax / resy are integer values.

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Make a RGB stack from 3 single-band input files

```
gdal raster stack red.tif green.tif blue.tif rgb.tif
```

Example 2: Make a virtual (VRT) stack from 2 single-band input files

```
gdal raster stack raster1.tif raster2.tif result.vrt
```

5.1.3.44 gdal raster tile

Added in version 3.11.

Synopsis

```
Usage: gdal raster tile [OPTIONS] <INPUT> <OUTPUT>

Generate tiles in separate files from a raster dataset.

Positional arguments:
  -i, --dataset, --input <INPUT>                                    Input raster datasets␣
↪[required] [not available in pipelines]
  -o, --output <OUTPUT>                                             Output directory␣
↪[required]

Common Options:
  -h, --help                                                        Display help message␣
↪and exit
  --json-usage                                                      Display usage as JSON␣
↪document and exit
  --config <KEY>=<VALUE>                                            Configuration option␣
↪[may be repeated]
  -q, --quiet                                                       Quiet mode (no␣
↪progress bar or warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>               Output format␣
↪(default: PNG)
  --co, --creation-option <KEY>=<VALUE>                             Creation option [may␣
↪be repeated]
  --tiling-scheme <TILING-SCHEME>                                   Tiling scheme. TILING-
↪SCHEME=raster|WebMercatorQuad|WorldCRS84Quad|WorldMercatorWGS84Quad|GoogleCRS84Quad|PseudoTMS_
↪GlobalMercator|GlobalGeodeticOriginLat270|LINZAntarticaMapTilegrid|APSTILE|CBMTILE|NZTM2000␣
↪(default: WebMercatorQuad)
  --min-zoom <MIN-ZOOM>                                             Minimum zoom level
  --max-zoom <MAX-ZOOM>                                             Maximum zoom level
```

(continues on next page)

(continued from previous page)

```
  --min-x <MIN-X>                                                Minimum tile X␣
↪coordinate
  --max-x <MAX-X>                                                Maximum tile X␣
↪coordinate
  --min-y <MIN-Y>                                                Minimum tile Y␣
↪coordinate
  --max-y <MAX-Y>                                                Maximum tile Y␣
↪coordinate
  --no-intersection-ok                                           Whether dataset␣
↪extent not intersecting tile matrix is only a warning
  -r, --resampling <RESAMPLING>                                  Resampling method for␣
↪max zoom.␣
↪RESAMPLING=nearest|bilinear|cubic|cubicspline|lanczos|average|rms|mode|min|max|med|q1|q3|sum␣
↪(default: cubic)
  --overview-resampling <OVERVIEW-RESAMPLING>                    Resampling method for␣
↪overviews. OVERVIEW-
↪RESAMPLING=nearest|bilinear|cubic|cubicspline|lanczos|average|rms|mode|min|max|med|q1|q3|sum
  --convention <CONVENTION>                                      Tile numbering␣
↪convention: xyz (from top) or tms (from bottom). CONVENTION=xyz|tms (default: xyz)
  --tile-size <TILE-SIZE>                                        Override default tile␣
↪size
  --add-alpha                                                    Whether to force␣
↪adding an alpha channel
                                                                 Mutually exclusive␣
↪with --no-alpha
  --no-alpha                                                     Whether to disable␣
↪adding an alpha channel
                                                                 Mutually exclusive␣
↪with --add-alpha
  --output-nodata <OUTPUT-NODATA>                                Output nodata value
  --skip-blank                                                   Do not generate blank␣
↪tiles
  --metadata <KEY>=<VALUE>                                       Add metadata item to␣
↪output tiles [may be repeated]
  --copy-src-metadata                                            Whether to copy␣
↪metadata from source dataset
  --aux-xml                                                      Generate .aux.xml␣
↪sidecar files when needed
  --kml                                                          Generate KML files
  --resume                                                       Generate only missing␣
↪files
  -j, --num-threads <NUM-THREADS>                                Number of jobs (or␣
↪ALL_CPUS) (default: ALL_CPUS)
  --parallel-method <PARALLEL-METHOD>                            Parallelization␣
↪method (thread, spawn, fork). PARALLEL-METHOD=thread|spawn|fork

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                            Input formats [may be␣
↪repeated] [not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                              Open options [may be␣
↪repeated] [not available in pipelines]
```

(continues on next page)

(continued from previous page)

```
Advanced Resampling Options:
  --excluded-values <EXCLUDED-VALUES>                              Tuples of values (e.g.
↪ <R>,<G>,<B> or (<R1>,<G1>,<B1>),(<R2>,<G2>,<B2>)) that must beignored as contributing␣
↪source pixels during (average) resampling
  --excluded-values-pct-threshold <EXCLUDED-VALUES-PCT-THRESHOLD>  Minimum percentage of␣
↪source pixels that must be set at one of the --excluded-values to cause the excluded␣
↪value to be used as the target pixel value (default: 50)
  --nodata-values-pct-threshold <NODATA-VALUES-PCT-THRESHOLD>      Minimum percentage of␣
↪source pixels that must be set at one of nodata (or alpha=0 or any other way to␣
↪express transparent pixelto cause the target pixel value to be transparent (default:␣
↪100)

Publication Options:
  --webviewer <WEBVIEWER>                                          Web viewer to␣
↪generate. WEBVIEWER=none|all|leaflet|openlayers|mapml|stac (default: all) [may be␣
↪repeated]
  --url <URL>                                                      URL address where the␣
↪generated tiles are going to be published
  --title <TITLE>                                                  Title of the map
  --copyright <COPYRIGHT>                                          Copyright for the map
  --mapml-template <MAPML-TEMPLATE>                                Filename of a␣
↪template mapml file where variables will be substituted
```

Description

gdal raster tile generates a directory with small tiles and metadata, following the OGC WMTS Specification. Simple web pages with viewers based on OpenLayers and Leaflet are generated as well - so anybody can comfortably explore your maps on-line and you do not need to install or configure any special software (like MapServer) and the map displays very fast in the web browser. You only need to upload the generated directory onto a web server.

It generates PNG files by default, but other output formats can be selected (JPEG, WEBP, GTiff, etc.). Note that not all formats support non-Byte data types. JPEG and WEBP only support Byte. PNG supports Byte and UInt16. GeoTiff can support all data types.

It can also use a tiling scheme fully adapted to the input raster, in terms of origin and resolution, when using the `raster` tiling scheme. In that scheme, tiles at the maximum zoom level will have the same resolution as the raster.

Starting with GDAL 3.12, **gdal raster tile** can be used as the last step of a pipeline.

Program-Specific Options

--aux-xml

Generate .aux.xml sidecar files when needed

--add-alpha

Whether to force adding an alpha channel. An alpha channel is added by default, unless the source dataset has a nodata value and the output format supports it.

--copy-src-metadata

Whether to copy metadata from source dataset into output tiles.

--convention `xyz|tms`

Tile numbering convention:

- `xyz` (default): from top, as in OGC Web Map Tiling Specification (WMTS)

- `tms`: from bottom, as in OSGeo Tile Map Service (TMS) Specification.

--output-nodata <NODATA>

Output nodata value.

-i, --input <INPUT>

Input raster dataset [required]

--kml

Generate Google Earth SuperOverlay metadata. Not compatible with tiling schemes with non-power-of-two zoom levels.

--max-x <MAX-X>

Maximum tile X coordinate, at maximum zoom level. Can be set to restrict the tiling process.

--max-y <MAX-Y>

Maximum tile Y coordinate, at maximum zoom level. Can be set to restrict the tiling process.

--max-zoom <MAX-ZOOM>

Maximum zoom level to generate. If not specified, this will be determined by comparing the resolution of the input dataset with the closest resolution in the list of tile matrix of the tile matrix set.

--metadata <KEY>=<VALUE>

Add metadata item to output tiles [may be repeated]

--min-x <MIN-X>

Minimum tile X coordinate, at maximum zoom level. Can be set to restrict the tiling process.

--min-y <MIN-Y>

Minimum tile Y coordinate, at maximum zoom level. Can be set to restrict the tiling process.

--min-zoom <MIN-ZOOM>

Minimum zoom level to generate. If not specified, equal to *--max-zoom*.

--no-alpha

Disable the creation of an alpha channel.

--no-intersection-ok

Whether the dataset extent not intersecting the tile matrix is only a warning. Otherwise, by default, an error will be emitted if that occurs.

-j, --num-threads <value>

Number of jobs to run at once. Default: number of CPUs detected.

-o, --output <OUTPUT>

Output directory [required]

The directory will be created if it does not exist, but its parent directory must exist.

--overview-resampling `nearest|bilinear|cubic|cubicspline|lanczos|average|rms|mode|min|max|med|q1|q3|sum`

Resampling method used to generate zoom levels lower than the maximum zoom level. Defaults to the value of *--resampling* is not specified.

--parallel-method `thread|spawn|fork`

Added in version GDAL: 3.12

Parallelization method:

- `thread` uses multi-threading, i.e. parallelized tasks are run within the same process.

- `spawn` launches child **gdal** sub-processes. `spawn` can achieve better "scaling", but requires the **gdal** binary to be available. GDAL will try to locate it, but you can also set the `GDAL_PATH` configuration option to point to the directory where the **gdal** binary is located.
- `fork` is a variant of `spawn`, using the system call `fork`, without executing the the **gdal** binary. Such method is not available on Windows. On Unix systems where it is available, this method is not recommended to be used on multithreaded processes, especially the ones where other threads are doing GDAL operation, since it can potentially cause deadlocks.

If *--parallel-method* is not specified, GDAL will automatically decide which of the method is the most appropriate, opting for `spawn` if the **gdal** binary can be located and a sufficient number of tiles per job are generated, and otherwise falling back to `fork` on Linux, MacOSX or FreeBSD (if no other thread is running), and otherwise to `thread`.

-r,
--resampling `nearest|bilinear|cubic|cubicspline|lanczos|average|rms|mode|min|max|med|q1|q3|sum`

Resampling method used to generate maximum zoom level, and also lower zoom levels if *--overview-resampling* is not specified. Defaults to `cubic`.

--resume

Generate only missing files. Can be used when interrupting a previous run to restart it.

--skip-blank

Do not generate fully blank/transparent tiles.

--tile-size `<PIXELS>`

Width and height of a tile, in pixels. Default is 256 for default tiling schemes. Setting it to a higher value enables generating higher DPI tile sets.

--tiling-scheme `<TILING-SCHEME>`

Tile cutting scheme. Defaults to WebMercatorQuad (also known as Google Maps compatible).

The following base profiles are available:

- `raster`: CRS = CRS of raster, tile size = 256 pixels, raster coverage at all zoom levels.
- `WebMercatorQuad`: CRS = EPSG:3857 ("WGS 84 / Pseudo-Mercator"), tile size: 256 pixels, whole world coverage at zoom level 0, one single tile at zoom level 0.
- `WorldCRS84Quad`: CRS = EPSG:4326 ("WGS 84"), tile size: 256 pixels, whole world coverage at zoom level 0, two side-by-side tiles at zoom level 0
- `WorldMercatorWGS84Quad`: CRS = EPSG:3395 ("WGS 84 / World Mercator"), tile size: 256 pixels, whole world coverage at zoom level 0, one single tile at zoom level 0. This can be seen as a variant of WebMercatorQuad except that it is fully conformal on the WGS84 ellipsoid.
- `GoogleCRS84Quad`: CRS = EPSG:4326 ("WGS 84"), tile size: 256 pixels, whole world coverage at zoom level 0, one single tile at zoom level 0 whose first and last 64 lines are blank (the top origin of the tile is a pseudo latitude of 180 degree, and its bottom -180 degree).
- `PseudoTMS_GlobalMercator`: CRS = EPSG:3857 ("WGS 84 / Pseudo-Mercator"), tile size: 256 pixels, whole world coverage at zoom level 0, 2x2 tiles at zoom level 0. This is equivalent to `WebMercatorQuad`, but with the zoom level shifted by one.
- `LINZAntarticaMapTileGrid`: LINZ Antarctic Map Tile Grid (Ross Sea Region). See tms_LINZAntarticaMapTileGrid.json
- `APSTILE`: Alaska Polar Stereographic-based tiled coordinate reference system for the Arctic region. See tms_MapML_APSTILE.json

- `CBMTILE`: Lambert Conformal Conic-based tiled coordinate reference system for Canada. See tms_MapML_CBMTILE.json
- `NZTM2000`: LINZ NZTM2000 Map Tile Grid. See tms_NZTM2000.json

Additional tiling schemes are discovered from `tms_XXXX.json` files placed in the GDAL data directory.

Advanced Resampling Options

--excluded-values=<EXCLUDED-VALUES>

Comma-separated tuple of values (thus typically "R,G,B"), that are ignored as contributing source pixels during resampling. The number of values in the tuple must be the same as the number of bands, excluding the alpha band. Several tuples of excluded values may be specified using the "(R1,G1,B2),(R2,G2,B2)" syntax. Only taken into account for average resampling. This concept is a bit similar to nodata/alpha, but the main difference is that pixels matching one of the excluded value tuples are still considered as valid, when determining the target pixel validity/density.

Added in version 3.11.1.

--excluded-values-pct-threshold=<EXCLUDED-VALUES-PCT-THRESHOLD>

Minimum percentage of source pixels that must be set at one of the --excluded-values to cause the excluded value, that is in majority among source pixels, to be used as the target pixel value. Default value is 50(%)

Added in version 3.11.1.

--nodata-values-pct-threshold=<NODATA-VALUES-PCT-THRESHOLD>

Minimum percentage of source pixels that must be at nodata (or alpha=0 or any other way to express transparent pixel) to cause the target pixel value to be transparent. Default value is 100 (%), which means that a target pixel is transparent only if all contributing source pixels are transparent. Only taken into account for average resampling.

Added in version 3.11.1.

Publication Options

--webviewer none|all|leaflet|openlayers|mapml|stac

Web viewer to generate. Defaults to `all`. Those web viewers are created at the root of the output directory.

`stac` generates a `stacta.json` file, following the Spatio-Temporal Asset Catalog Tiled Assets specification, and that can be opened by the *STACTA driver*.

--url

URL address where the generated tiles are going to be published.

--title <TITLE>

Title of the map.

--copyright <COPYRIGHT>

Copyright for the map.

MapML options

MapML webviewer can only be generated if *--convention* is set to the default value `xyz`.

The following profiles are supported:

- `WebMercatorQuad`: mapped to OSMTILE MapML tiling scheme
- `GoogleCRS84Quad`: mapped to WGS84 MapML tiling scheme
- `APSTILE`: from the tms_MapML_APSTILE.json data file

- CBMTILE: from the tms_MapML_CBMTILE.json data file

The generated MapML file in the output directory is `mapml.mapl`

Available options are:

--mapml-template <filename>

Filename of a template mapml file where variables will be substituted. If not specified, the generic template_tiles.mapml file from GDAL data resources will be used

The *--url* option is also used to substitute ${URL} in the template MapML file.

Standard Options

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

Defaults to PNG.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Generate PNG tiles with WebMercatorQuad tiling scheme for zoom levels 2 to 5.

```
gdal raster tile --min-zoom=2 --max-zoom=5 input.tif output_folder
```

Example 2: Retile a raster using its origin and resolution

```
gdal raster tile --tiling-scheme raster input.tif output_folder
```

Example 3: Creating a tiled dataset, compatible with the Spatio-Temporal Asset Catalog Tiled Assets specification, using Cloud-Optimized GeoTIFF metatiles of dimension 4096x4096.

```
gdal raster tile --format COG --tile-size 4096 input.tif output_folder
```

Example 4: Mosaic on-the-fly several input files and tile that mosaic.

```
gdal raster pipeline ! mosaic input*.tif ! tile output_folder
```

5.1.3.45 gdal raster tpi

Added in version 3.11.

Synopsis

```
Usage: gdal raster tpi [OPTIONS] <INPUT> <OUTPUT>

Generate a Topographic Position Index (TPI) map

Positional arguments:
  -i, --input <INPUT>                                    Input raster datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output raster dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                         Mutually exclusive with --append
  --append                                               Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                         Mutually exclusive with --
↪overwrite
  -b, --band <BAND>                                      Input band (1-based index)␣
```

(continues on next page)

(continued from previous page)

```
↪(default: 1)
  --no-edges                                        Do not try to interpolate values␣
↪at dataset edges or close to nodata values

Advanced Options:
  --if, --input-format <INPUT-FORMAT>               Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                 Open options [may be repeated]␣
↪[not available in pipelines]
```

Description

gdal raster tpi generates a single-band raster with values computed from the elevation. TPI stands for Topographic Position Index, which is defined as the difference between a central pixel and the mean of its surrounding cells (see Wilson et al 2007, Marine Geodesy 30:3-35).

This subcommand is also available as a potential step of *gdal raster pipeline*

Value -9999 is used as the nodata value.

A nodata value in the target dataset will also be emitted if at least one pixel set to the nodata value is found in the 3x3 window centered around each source pixel. By default, the algorithm will compute values at image edges or if a nodata value is found in the 3x3 window, by interpolating missing values, unless `--no-edges` is specified, in which case a 1-pixel border around the image will be set with the nodata value.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

-b, --band <BAND>

Index (starting at 1) of the band to which the TPI must be computed.

--no-edges

Do not try to interpolate values at dataset edges or close to nodata values

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation

for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Generates a Topographic Position Index (TPI) map from a DTED0 file.

```
$ gdal raster tpi n43.dt0 out.tif --overwrite
```

5.1.3.46 gdal raster tri

Added in version 3.11.

Synopsis

```
Usage: gdal raster tri [OPTIONS] <INPUT> <OUTPUT>

Generate a Terrain Ruggedness Index (TRI) map

Positional arguments:
  -i, --input <INPUT>                                    Input raster datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output raster dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
```

(continues on next page)

(continued from previous page)

```
  --config <KEY>=<VALUE>                                Configuration option [may be␣
↪repeated]
  -q, --quiet                                           Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                 Creation option [may be repeated]␣
↪[not available in pipelines]
  --overwrite                                           Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                        Mutually exclusive with --append
  --append                                              Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                        Mutually exclusive with --
↪overwrite
  -b, --band <BAND>                                     Input band (1-based index)␣
↪(default: 1)
  --algorithm <ALGORITHM>                               Algorithm to compute TRI.␣
↪ALGORITHM=Riley|Wilson (default: Riley)
  --no-edges                                            Do not try to interpolate values␣
↪at dataset edges or close to nodata values

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                   Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                     Open options [may be repeated]␣
↪[not available in pipelines]
```

Description

gdal raster tri generates a single-band raster with values computed from the elevation. TRI stands for Terrain Ruggedness Index, which measures the difference between a central pixel and its surrounding cells.

This subcommand is also available as a potential step of *gdal raster pipeline*

Value -9999 is used as the nodata value.

A nodata value in the target dataset will also be emitted if at least one pixel set to the nodata value is found in the 3x3 window centered around each source pixel. By default, the algorithm will compute values at image edges or if a nodata value is found in the 3x3 window, by interpolating missing values, unless `--no-edges` is specified, in which case a 1-pixel border around the image will be set with the nodata value.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

--algorithm `Riley|Wilson`

Select the algorithm to use:

- `Riley`, the default, uses the square root of the sum of the square of the difference between a central pixel and its surrounding cells. This is recommended for terrestrial use cases. It is implemented from Riley, S.J., De Gloria, S.D., Elliot, R. (1999): A Terrain Ruggedness that Quantifies Topographic Heterogeneity. Intermountain Journal of Science, Vol.5, No.1-4, pp.23-27
- `Wilson` uses the mean difference between a central pixel and its surrounding cells. This is recommended for bathymetric use cases. It is implemented from Wilson et al. 2007, Marine Geodesy 30:3-35.

-b, --band `<BAND>`

Index (starting at 1) of the band to which the TRI must be computed.

--no-edges

Do not try to interpolate values at dataset edges or close to nodata values

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option `<NAME>=<VALUE>`

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format `<format>`

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option `<NAME>=<VALUE>`

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format `<OUTPUT-FORMAT>`

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Generates a Terrain Ruggedness Index (TRI) map from a DTED0 file.

```
$ gdal raster tri n43.dt0 out.tif --overwrite
```

5.1.3.47 gdal raster unscale

Added in version 3.11.

Synopsis

```
Usage: gdal raster unscale [OPTIONS] <INPUT> <OUTPUT>

Convert scaled values of a raster dataset into unscaled values.

Positional arguments:
  -i, --input <INPUT>                                       Input raster datasets␣
↪[required] [not available in pipelines]
  -o, --output <OUTPUT>                                     Output raster dataset␣
↪[required] [not available in pipelines]

Common Options:
  -h, --help                                                Display help message and exit
  --json-usage                                              Display usage as JSON␣
↪document and exit
  --config <KEY>=<VALUE>                                    Configuration option [may be␣
↪repeated]
  -q, --quiet                                               Quiet mode (no progress bar␣
↪or warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>       Output format ("GDALG"␣
↪allowed) [not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                     Creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                               Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                            Mutually exclusive with --
↪append
  --append                                                  Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                            Mutually exclusive with --
↪overwrite
  --ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>  Output data type. OUTPUT-DATA-
↪TYPE=UInt8|Int8|UInt16|Int16|UInt32|Int32|UInt64|Int64|CInt16|CInt32|Float16|Float32|Float64|CFloat32

Advanced Options:
```

(continues on next page)

(continued from previous page)

```
  --if, --input-format <INPUT-FORMAT>                    Input formats [may be␣
↪repeated] [not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                      Open options [may be␣
↪repeated] [not available in pipelines]
```

Description

gdal raster unscale applies the scale/offset metadata for the bands to convert scaled values to unscaled values.

If the input band data type is Byte, Int8, UInt16, Int16, UInt32, Int32, Int64, UInt64, Float16 or Float32, the default output data type will be Float32. If the input band data type is Float64, it will be kept as the default output data type. If the input band data type is CInt16, CFloat16 or CFloat32, the default output data type will be Float32. If the input band data type is CFloat64, it will be kept as the default output data type.

The unscaled value is computed from the scaled raw value with the following formula:

$$unscaled_value = scaled_value * scale + offset$$

If one of the input bands has no scale/offset metadata, its values are kept unmodified.

This command is the reverse operation of *gdal raster scale*.

This subcommand is also available as a potential step of *gdal raster pipeline*

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

--ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>

Output data type among `Byte`, `Int8`, `UInt16`, `Int16`, `UInt32`, `Int32`, `UInt64`, `Int64`, `CInt16`, `CInt32`, `Float32`, `Float64`, `CFloat32`, `CFloat64`.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Unscale a scaled raster to a Float32 one

```
$ gdal raster unscale scaled_byte.tif unscaled_float32.tif --overwrite
```

5.1.3.48 gdal raster update

Added in version 3.12.

Synopsis

```
Usage: gdal raster update [OPTIONS] <INPUT> <OUTPUT>

Update the destination raster with the content of the input one.

Positional arguments:
  -i, --input <INPUT>                       Input raster datasets [required] [not␣
↪available in pipelines]
  -o, --output <OUTPUT>                     Output raster dataset [required]

Common Options:
  -h, --help                                Display help message and exit
  --json-usage                              Display usage as JSON document and exit
  --config <KEY>=<VALUE>                    Configuration option [may be repeated]
  -q, --quiet                               Quiet mode (no progress bar or warning␣
↪message) [not available in pipelines]

Options:
  --geometry <GEOMETRY>                     Clipping geometry (WKT or GeoJSON)
  --geometry-crs <GEOMETRY-CRS>             CRS of clipping geometry
  -r, --resampling <RESAMPLING>             Resampling method.␣
```

(continues on next page)

(continued from previous page)

```
↪RESAMPLING=nearest|bilinear|cubic|cubicspline|lanczos|average|rms|mode|min|max|med|q1|q3|sum␣
↪(default: nearest)
  --no-update-overviews                      Do not update existing overviews

Advanced Options:
  --if, --input-format <INPUT-FORMAT>        Input formats [may be repeated] [not␣
↪available in pipelines]
  --oo, --open-option <KEY>=<VALUE>          Open options [may be repeated] [not␣
↪available in pipelines]
  --wo, --warp-option <NAME>=<VALUE>         Warping option(s) [may be repeated]
  --to, --transform-option <NAME>=<VALUE>    Transform option(s) [may be repeated]
  --et, --error-threshold <ERROR-THRESHOLD>  Error threshold
```

Description

gdal raster update can be used to update an existing output raster dataset with the pixel values of the input raster, doing reprojection if the input and output datasets do not have the same coordinate reference systems. The extent, size, resolution or coordinate reference system of the output dataset are not modified by this operation.

Overviews are updated by this command (using *gdal raster overview refresh*), unless `--no-update-overviews` is specified.

Starting with GDAL 3.13, **gdal raster update** can be used as a step of a pipeline, with the input dataset being the output of the previous step.

Program-Specific Options

--et, --error-threshold <ERROR-THRESHOLD>

Error threshold for transformation approximation, expressed as a number of input pixels. Defaults to 0.125 pixels unless the `RPC_DEM` transformer option is specified, in which case an exact transformer, i.e. `--error-threshold=0`, will be used.

--geometry <WKT_or_GeoJSON>

Geometry as a WKT or GeoJSON string of a polygon (or multipolygon) to which to restrict the update. If the input geometry is GeoJSON, its CRS is assumed to be WGS84, unless there is a CRS defined in the GeoJSON geometry or `--geometry-crs` is specified. If the input geometry is WKT, its CRS is assumed to be the one of the input dataset, unless `--geometry-crs` is specified. The X and Y axis are the "GIS friendly ones", that is X is longitude or easting, and Y is latitude or northing.

--geometry-crs <CRS>

CRS in which the coordinate values of `--geometry` are expressed. If not specified, it is assumed to be the CRS of the input dataset. The geometry will be reprojected from the geometry-crs to the CRS of the dataset being updated.

--no-update-overviews

Do not update existing overviews.

--to, --transform-option <NAME>=<VALUE>

Set a transformer option suitable to pass to *GDALCreateGenImgProjTransformer2()*. See *GDALCreateRPCTransformerV2()* for RPC specific options.

-r, --resampling <RESAMPLING>

Resampling method to use. Available methods are:

`nearest`: nearest neighbour resampling (default, fastest algorithm, worst interpolation quality).

`bilinear`: bilinear resampling.

`cubic`: cubic resampling.

`cubicspline`: cubic spline resampling.

`lanczos`: Lanczos windowed sinc resampling.

`average`: average resampling, computes the weighted average of all non-NODATA contributing pixels.

`rms` root mean square / quadratic mean of all non-NODATA contributing pixels

`mode`: mode resampling, selects the value which appears most often of all the sampled points. In the case of ties, the first value identified as the mode will be selected.

`max`: maximum resampling, selects the maximum value from all non-NODATA contributing pixels.

`min`: minimum resampling, selects the minimum value from all non-NODATA contributing pixels.

`med`: median resampling, selects the median value of all non-NODATA contributing pixels.

`q1`: first quartile resampling, selects the first quartile value of all non-NODATA contributing pixels.

`q3`: third quartile resampling, selects the third quartile value of all non-NODATA contributing pixels.

`sum`: compute the weighted sum of all non-NODATA contributing pixels

--wo, --warp-option <NAME>=<VALUE>

Set a warp option. The *GDALWarpOptions::papszWarpOptions* docs show all options. Multiple options may be listed.

Program-Specific Options

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Update existing out.tif with content of in.tif using cubic interpolation

```
$ gdal raster update -r cubic in.tif out.tif
```

5.1.3.49 gdal raster viewshed

Added in version 3.11.

Synopsis

```
Usage: gdal raster viewshed [OPTIONS] <INPUT> <OUTPUT>

Compute the viewshed of a raster dataset.

Positional arguments:
  -i, --input <INPUT>                                    Input raster datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output raster dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
                                                         Mutually exclusive with --append
  --append                                               Append as a subdataset to␣
↪existing output [not available in pipelines]
                                                         Mutually exclusive with --
↪overwrite
  -p, --pos, --position <X,Y> or <X,Y,H>                 Observer position [2..3 values]
  -z, --height <HEIGHT>                                  Observer height
  --sd-filename <SD-FILENAME>                            Filename of standard-deviation␣
↪raster
  --target-height <TARGET-HEIGHT>                        Height of the target above the␣
↪DEM surface in the height unit of the DEM. (default: 0)
  --mode <MODE>                                          Sets what information the output␣
↪contains.. MODE=normal|DEM|ground|cumulative (default: normal)
  --max-distance <MAX-DISTANCE>                          Maximum distance from observer to␣
↪compute visibility. It is also used to clamp the extent of the output raster.
  --min-distance <MIN-DISTANCE>                          Mask all cells less than this␣
↪distance from the observer. Must be less than 'max-distance'.
  --start-angle <START-ANGLE>                            Mask all cells outside of the arc␣
↪('start-angle', 'end-angle'). Clockwise degrees from north. Also used to clamp the␣
↪extent of the output raster.
  --end-angle <END-ANGLE>                                Mask all cells outside of the arc␣
```

(continues on next page)

(continued from previous page)

```
↪('start-angle', 'end-angle'). Clockwise degrees from north. Also used to clamp the␣
↪extent of the output raster.
  --high-pitch <HIGH-PITCH>                              Mark all cells out-of-range where␣
↪the observable height would be higher than the 'high-pitch' angle from the observer.␣
↪Degrees from horizontal - positive is up. Must be greater than 'low-pitch'.
  --low-pitch <LOW-PITCH>                                Bound observable height to be no␣
↪lower than the 'low-pitch' angle from the observer. Degrees from horizontal - positive␣
↪is up. Must be less than 'high-pitch'.
  --curvature-coefficient <CURVATURE-COEFFICIENT>        Coefficient to consider the␣
↪effect of the curvature and refraction.
  -b, --band <BAND>                                      Input band (1-based index)␣
↪(default: 1)
  --visible-value <VISIBLE-VALUE>                        Pixel value to set for visible␣
↪areas (default: 255)
  --invisible-value <INVISIBLE-VALUE>                    Pixel value to set for invisible␣
↪areas (default: 0)
  --maybe-visible-value <MAYBE-VISIBLE-VALUE>            Pixel value to set for␣
↪potentially visible areas (default: 2)
  --out-of-range-value <OUT-OF-RANGE-VALUE>              Pixel value to set for the cells␣
↪that fall outside of the range specified by the observer location and the maximum␣
↪distance (default: 0)
  --output-nodata <OUTPUT-NODATA>                        The value to be set for the cells␣
↪in the output raster that have no data.
  --observer-spacing <OBSERVER-SPACING>                  Cell Spacing between observers␣
↪(default: 10)
  -j, --num-threads <NUM-THREADS>                        Number of jobs (or ALL_CPUS)␣
↪(default: 3)

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                    Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                      Open options [may be repeated]␣
↪[not available in pipelines]
```

Description

gdal raster viewshed creates a binary visibility raster from one band of the input raster elevation model (DEM). The output raster will be of type Byte. Using the `DEM` or `ground` values of `--mode` can also create a minimum visible height raster of type Float64.

It uses the method defined in [Wang2000] for a user defined point.

This subcommand is also available as a potential step of *gdal raster pipeline* (since GDAL 3.12)

Added in version 3.12.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

> **Note**
>
> However this algorithm is not natively streaming compatible. Consequently a temporary dataset will be generated, which may cause significant processing time at opening.

Program-Specific Options

-b, --band <band>

Select an input band containing the DEM data. Bands are numbered from 1. Only a single band can be used. Only the part of the raster within the specified maximum distance around the observer point is processed.

--curvature-coefficient <value>

Coefficient to consider the effect of the curvature and refraction. When calculating visibility between two points (i.e. Line Of Sight or Viewshed), The magnitude of this effect varies with atmospheric conditions and depends on the wavelength.

Different applications for calculating visibility use different interchangeable notation to describe this phenomena: Refraction Coefficient, Curvature Coefficient, and Sphere Diameter Factor. gdal_viewshed uses the Curvature Coefficient notation.

$$CurvCoeff = 1 - RefractionCoeff$$

Changes in air density curve the light downward causing an observer to see further and the earth to appear less curved, as if the sphere (earth) diameter is larger then it actually is. The ratio between that imaginary sphere diameter and the actual sphere diameter is given by the formula:

$$SphereDiameterFactor = 1/CurvCoeff = 1/(1 - RefractionCoeff)$$

For visible light, the standard atmospheric refraction coefficient that is generally used is 1/7. Thus the default value (since GDAL 3.4) for CurvCoeff that gdal_viewshed uses is 0.85714 (=~ 1-1/7) for Earth CRS. Starting with GDAL 3.6, for non-Earth CRS (those whole semi-major axis differs by more than 5% with the one of WGS 84), CurvCoeff default value is 1.0, to account for the no refraction use case.

The height of the DEM is corrected according to the following formula:

$$Height_{Corrected} = Height_{DEM} - CurvCoeff \frac{TargetDistance^2}{SphereDiameter}$$

Typical coefficient values are given in the table below (use Curvature Coeff value for the cc option)

Use Case	Refraction Coeff	**Curvature Coeff**	Sphere Diameter Factor
No Refraction	0	1	1
Visible Light	1/7	6/7 (=~0.85714)	7/6 (=~1.1666)
Radio Waves	0.25 ~ 0.325	0.75 ~ 0.675	1.33 ~ 1.48
Flat Earth	1	0	inf

--end-angle <value>

Added in version 3.12.

End angle for visibility. Measured clockwise from 0 North, in degree. (Not supported in cumulative mode)

-z, --height <HEIGHT>

Observer height

--high-pitch <value>

Added in version 3.12.

High angle for visibility. Measured up from 0 horizontal, in degree. Input values above the high pitch are marked out of range. Must be greater than '--low-pitch'. (Not supported in cumulative mode)

--invisible-value <value>

Pixel value to set for invisible areas. (Not supported in cumulative mode) Default: 0

--low-pitch <value>

Added in version 3.12.

Low angle for visibility. Measured up from 0 horizontal, in degree. Input cell values below the pitch are are clamped to be no lower than the intersection of the angle. Must be less than '--high-pitch'. (Not supported in cumulative mode)

--max-distance <value>

Maximum distance from observer to compute visibility. It is also used to clamp the extent of the output raster. (Not supported in cumulative mode)

--min-distance <value>

Added in version 3.12.

Minimum distance from observer to compute visibility. Must be less than '--max-distance' (Not supported in cumulative mode)

--mode normal|DEM|ground|cumulative

Sets what information the output contains.

- `normal` (the default) returns a raster of type Byte containing visible locations.
- `DEM` and `ground` return a raster of type Float64 containing the minimum target height for the target to be visible from the DEM surface or ground level respectively. That is to say, if the minimum target height for the target to be visible at a point is `h` and the value of the input raster at that point is `E`, for `DEM`, `E + h` will be the output value. For `ground`, `h` will be output value. Options `--target-height`, `--invisible-value` and `--visible-value` will be ignored.
- `cumulative` creates an eight bit raster the same size as the input raster where each cell represents the relative observability from a grid of observer points. See the *`--observer-spacing`* option.

-j, --num-threads <value>

Number of jobs to run at once. (only supported in cumulative mode). Default: 3

--observer-spacing <value>

Cell spacing between observers (only supported in cumulative mode). Default: 10

--out-of-range-value <value>

Pixel value to set for the cells that fall outside of the range specified by the observer location and the maximum distance. (Not supported in cumulative mode) Default: 0

--output-nodata <value>

The value to be set for the cells in the output raster that have no data.

> **Note**
>
> Currently, no special processing of input cells at a nodata value is done (which may result in erroneous results).

-p, --pos, --position <X,Y> or <X,Y,H>

The X,Y or X,Y,H(Height) position of the observer (in SRS units for X and Y, and in the height unit of the DEM for H). If the coordinate is outside of the raster, all space between the observer and the raster is assumed not to occlude visibility of the raster. (Not supported in cumulative mode.) If H is not specified, it defaults to 2.

--start-angle <value>

Added in version 3.12.

Start angle for visibility. Measured clockwise from 0 North, in degree. (Not supported in cumulative mode)

--target-height <value>

The height of the target above the DEM surface in the height unit of the DEM. Default: 0

--visible-value <value>

Pixel value to set for visible areas. (Not supported in cumulative mode) Default: 255

Added in version 3.13.

--sd-filename <value>

Filename of raster containing standard deviations of the input raster values. The raster always comes from band 1 and the size must match that of the input raster.

--maybe-visible-value <value>

Pixel value to set for visible areas. (Not supported in cumulative mode) Default: 255

Standard Options

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

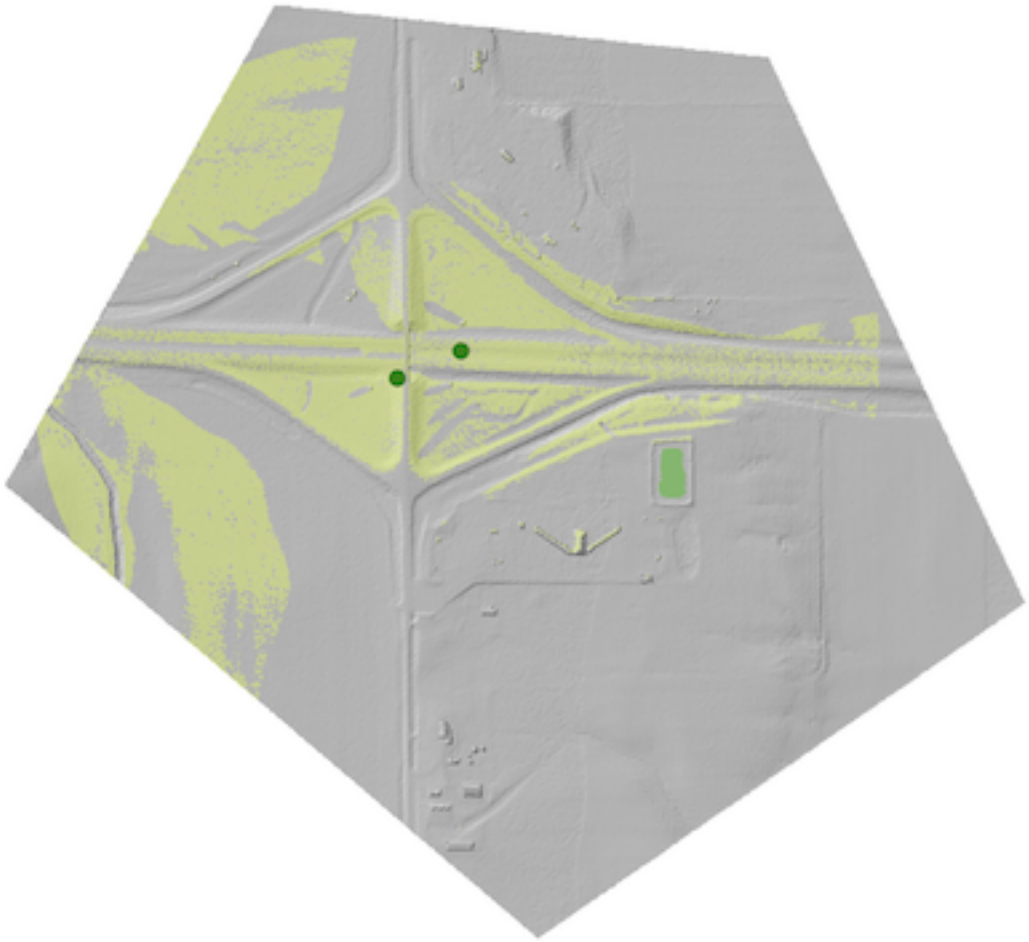
Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1

Screenshot of 2 combined viewshed analysis, with the yellow pixels showing the area that is visible from the both observation locations (the green dots), while the small green area is only visible from one location.



Create a viewshed raster with a radius of 500 for a person standing at location (-10147017, 5108065).

```
gdal raster viewshed --max-distance=500 --pos=-10147017,5108065 source.tif destination.
↪tif
```

5.1.3.50 gdal raster pipeline write

Added in version 3.11.

Description

The `write` operation is for use in a *gdal pipeline* only, and writes a raster dataset. This is the last step of a pipeline.

To write a temporary dataset in the middle of a pipeline, use *gdal raster materialize*.

Synopsis

```
* write [OPTIONS] <OUTPUT>
--------------------------

Write a raster dataset.

Positional arguments:
  -o, --output <OUTPUT>                                Output raster dataset [required]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)
  --co, --creation-option <KEY>=<VALUE>                Creation option [may be repeated]
  --overwrite                                          Whether overwriting existing␣
↪output dataset is allowed
                                                       Mutually exclusive with --append
  --append                                             Append as a subdataset to␣
↪existing output
                                                       Mutually exclusive with --
↪overwrite
```

Standard Options

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--append

Append input raster as a new subdataset to an existing output file. Only works with drivers that support adding subdatasets such as *GTiff -- GeoTIFF File Format* and *GPKG -- GeoPackage raster* This also creates the output dataset if it does not exist yet.

Examples

Example 1: Write a GeoTIFF file

```
$ gdal raster pipeline ... [other commands here] ... ! write out.tif --overwrite
```

5.1.3.51 gdal raster zonal-stats

Added in version 3.12.

Synopsis

```
Usage: gdal raster zonal-stats [OPTIONS] <INPUT> <OUTPUT>

Calculate raster zonal statistics

Positional arguments:
  -i, --input <INPUT>                                    Input raster datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>           Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                               Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                      Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
```

(continues on next page)

(continued from previous page)

```
  --append                                                Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                          Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                           Output layer name [not available␣
↪in pipelines]
  --skip-errors                                           Skip errors when writing features␣
↪[not available in pipelines]
  -b, --band <BAND>                                       Input band(s) (1-based index)␣
↪[may be repeated]
  --zones <ZONES>                                         Dataset containing zone␣
↪definitions [required]
  --zones-band <ZONES-BAND>                               Band from which zones should be␣
↪read
                                                          Mutually exclusive with --zones-
↪layer
  --zones-layer <ZONES-LAYER>                             Layer from which zones should be␣
↪read
                                                          Mutually exclusive with --zones-
↪band
  --weights <WEIGHTS>                                     Weighting raster dataset
  --weights-band <WEIGHTS-BAND>                           Band from which weights should be␣
↪read (default: 1)
  --pixels <PIXELS>                                       Method to determine which pixels␣
↪are included in stat calculation.. PIXELS=default|fractional|all-touched
  --stat <STAT>                                           Statistic(s) to compute for each␣
↪zone. STAT=center_x|center_y|count|coverage|frac|max|max_center_x|max_center_
↪y|mean|median|min|minority|min_center_x|min_center_
↪y|mode|stdev|sum|unique|values|variance|variety|weighted_mean|weighted_stdev|weighted_
↪sum|weighted_variance|weights [may be repeated] [required]
  --include-field <INCLUDE-FIELD>                         Fields from polygon zones to␣
↪include in output [may be repeated]
  --include-geom                                          Include polygon zone geometry in␣
↪the output
  --strategy <STRATEGY>                                   For polygon zones, whether to␣
↪iterate over input features or raster chunks. STRATEGY=feature|raster (default:␣
↪feature)
  --chunk-size <CHUNK-SIZE>                               Maximum size of raster chunks␣
↪read into memory (default: 5%)

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                     Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                       Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>         Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                                Upsert features (implies 'append
↪') [not available in pipelines]
                                                          Mutually exclusive with --append
```

Description

gdal raster zonal-stats computes raster zonal statistics -- a summary of pixel values within zones specified either polygon features or a categorical raster.

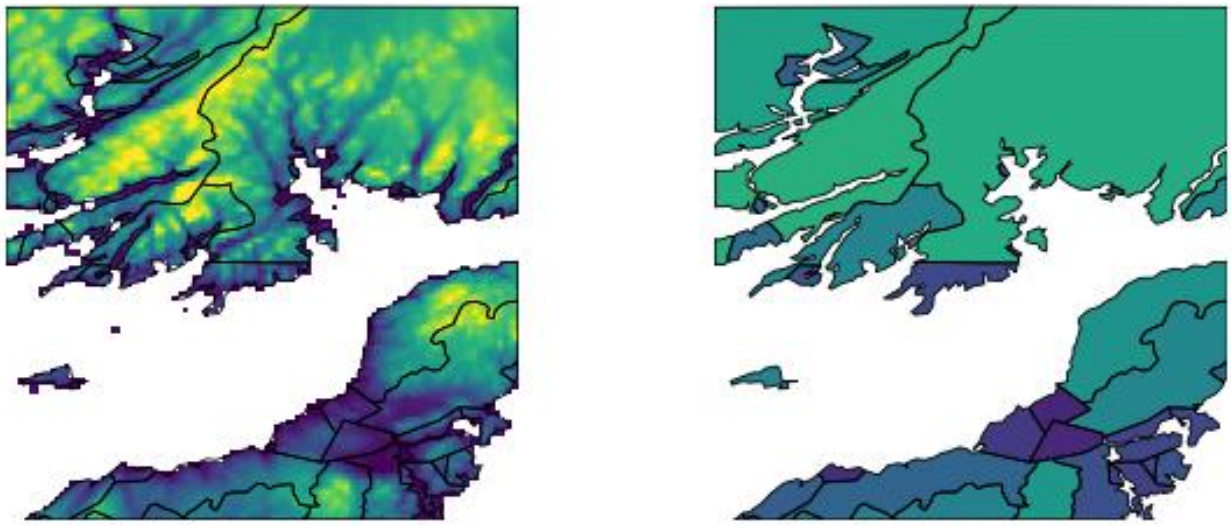


Fig. 4: Mean elevation calculated for each polygon from an elevation raster.

A weighting raster may be provided in addition to the source raster, allowing calculation of statistics such as population-weighted mean weather conditions.

All inputs should be provided in the same spatial reference system. If input rasters use different resolutions they will be resampled to a consistent resolution using average resampling. However, no reprojection is performed.

Pixel inclusion and weighting

When zones are defined using polygons, three methods are available for determining which pixels to include in the calculation. By default, the geometries are rasterized using default parameters, and pixels included in the rasterization are used in the calculation with equal weighting. By using `--pixels all-touched`, all pixels touched by the polygon are used in the calculation, again with equal weighting. If using `--pixels fractional`, the fraction of each pixel that is covered by the polygon is computed, and pixels are weighted according to this fraction. This option requires GDAL be built against version 3.14 or later of the GEOS library.

Supported stats

Name	Description
center_x	Array with cell center x-coordinate for each cell that intersects the polygon. Each cell center may or may not be inside the polygon.
center_y	Array with cell center y-coordinate for each cell that intersects the polygon. Each cell center may or may not be inside the polygon.
count	Sum of all cell coverage fractions.
coverage	Array with coverage fraction of each cell that intersects the polygon
frac	Fraction of covered cells that are occupied by each distinct raster value, as provided by `unique`.
max	Maximum value of cells that intersect the polygon, not taking coverage fractions or weighting raster values into account.
max_center_	Cell center x-coordinate for the cell containing the maximum value intersected by the polygon. The center of this cell may or may not be inside the polygon.
max_center_	Cell center y-coordinate for the cell containing the maximum value intersected by the polygon. The center of this cell may or may not be inside the polygon.
mean	Mean value of cells that intersect the polygon, weighted by the percent of each cell that is covered.
min	Minimum value of cells that intersect the polygon, not taking coverage fractions or weighting raster values into account.
min_center_	Cell center x-coordinate for the cell containing the minimum value intersected by the polygon. The center of this cell may or may not be inside the polygon.
min_center_	Cell center y-coordinate for the cell containing the minimum value intersected by the polygon. The center of this cell may or may not be inside the polygon.
minority	The raster value occupying the least number of cells, taking into account cell coverage fractions but not weighting raster values.
mode	The raster value occupying the greatest number of cells, taking into account cell coverage fractions but not weighting raster values.
stdev	Population standard deviation of cell values that intersect the polygon, taking into account coverage fraction.
sum	Sum of values of raster cells that intersect the polygon, with each raster value weighted by its coverage fraction.
unique	Array of unique raster values for cells that intersect the polygon
values	Array of raster values for each cell that intersects the polygon
variance	Population variance of cell values that intersect the polygon, taking into account coverage fraction.
variety	The number of distinct raster values in cells wholly or partially covered by the polygon.
weighted_fr	Fraction of covered cells that are occupied by each distinct raster value, weighted by the value of a second weighting raster. Order corresponds to values returned by `unique`.
weighted_m	Mean value of cells that intersect the polygon, weighted by the product over the coverage fraction and the weighting raster.
weighted_st	Weighted version of `stdev`.
weighted_va	Weighted version of `variance`
weights	Array of weight values for each cell that intersects the polygon

This algorithm can be part of a *gdal pipeline*.

Program-Specific Options

--band, -b <BAND>

Input band(s) to be processed.

--chunk-size <MEMORY>

Defines the maximum size of raster chunks to read. May be expressed as either an amount of memory (500 MB, 2 GB, etc.) or as a percentage of available RAM (e.g. 10%).

--include-field <INCLUDE-FIELD>

Specifies one or more fields from the zones to be copied to the output. Only available when vector zones are used. Since GDAL 3.13, the special values "ALL" and "NONE" can be used.

--include-geom

Include the zone geometry in the output feature. Only available when vector zones are used.

Added in version 3.13.

--pixels <PIXELS>

Method to determine which pixels should be included in the calculation: `default`, `all-touched`, or `fractional`.

--stat <STAT>

Specifies one or more of the *Supported stats* to compute for each zone.

--strategy <STRATEGY>

Specifies the the processing strategy (`raster` or `feature`), when vector zones are used. In the default strategy (`--strategy feature`), GDAL will iterate over the features in the zone dataset, read the corresponding pixels from the raster, and write the statistics for that feature. This avoids the need to read the entire feature dataset into memory at once, but may cause the same pixels to be read multiple times if the polygon features are large or not ordered spatially. If `--strategy raster` is used, GDAL will iterate over chunks of the raster dataset, find corresponding polygon zones, and update the statistics for those features. (The size of the raster chunks can be controlled using :option:`--chunk-size`.) This ensures that raster pixels are only read once, but may cause the same features to be processed multiple times.

--weights <WEIGHTS>

Optional path to a dataset to use for weighting.

--weights-band <WEIGHTS-BAND>

Specifies the raster band from which weights can be read.

--zones <ZONES>

The path to a dataset that defines the zones. If the dataset has more than 1 raster band or feature layer, *--zones-band* or *--zones-layer* must be specified.

--zones-band <ZONES-BAND>

Specifies the raster band from which zones can be read.

--zones-layer <ZONES-LAYER>

Specifies the feature layer from which zones can be read.

Standard Options

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-layer <NAME>

Specifies the name of the layer to which features will be written. If not specified, output layer names will be the same as input layer names.

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Summarize mean elevation within 200m of points of interest

Using a nested pipeline for the zone dataset

```
  gdal pipeline read dem.tif ! \
      zonal-stats \
        --zones [ read points.geojson ! buffer 200 ] \
        --stat mean ! \
      write \
        --output-format CSV \
        --output /vsistdout/

or, using the zone vector dataset as the piped dataset using the ``_PIPE_`` placeholder␣
↪dataset name:
```

```
gdal pipeline read points.geojson ! \
    buffer 200 ! \
    zonal-stats \
      --input dem.tif
      --zones _PIPE_ \
      --stat mean ! \
    write \
      --output-format CSV \
      --output /vsistdout/
```

Example 2: Create a layer with the highest points in each watershed

```
gdal pipeline read dem.tif !
    zonal-stats \
```

(continues on next page)

(continued from previous page)

```
      --zones watersheds.shp \
      --stat max_center_x \
      --stat max_center_y ! \
    make-point \
      --x max_center_x \
      --y max_center_y \
      --output-crs EPSG:4326 ! \
    write out.geojson
```

5.1.4 Vector commands

5.1.4.1 gdal vector

Added in version 3.11.

Synopsis

```
Usage: gdal vector <SUBCOMMAND> [OPTIONS]
where <SUBCOMMAND> is one of:
  - buffer:              Compute a buffer around geometries of a vector dataset.
  - check-coverage:      Check a polygon coverage for validity
  - check-geometry:      Check a dataset for invalid geometries
  - clean-coverage:      Alter polygon boundaries to make shared edges identical,␣
↪removing gaps and overlaps
  - clip:                Clip a vector dataset.
  - combine:             Combine features into collections
  - concat:              Concatenate vector datasets.
  - concave-hull:        Compute the concave hull of geometries of a vector dataset.
  - convert:             Convert a vector dataset.
  - convex-hull:         Compute the convex hull of geometries of a vector dataset.
  - create:              Create a vector dataset.
  - dissolve:            Dissolves multipart features
  - edit:                Edit metadata of a vector dataset.
  - explode-collections: Explode geometries of type collection of a vector dataset.
  - export-schema:       Export the OGR_SCHEMA from a vector dataset.
  - filter:              Filter a vector dataset.
  - grid:                Create a regular grid from scattered points.
  - index:               Create a vector index of vector datasets.
  - info:                Return information on a vector dataset.
  - layer-algebra:       Perform algebraic operation between 2 layers.
  - make-point:          Create point geometries from attribute fields
  - make-valid:          Fix validity of geometries of a vector dataset.
  - partition:           Partition a vector dataset into multiple files.
  - pipeline:            Process a vector dataset applying several steps.
  - rasterize:           Burns vector geometries into a raster.
  - rename-layer:        Rename layer(s) of a vector dataset.
  - reproject:           Reproject a vector dataset.
  - segmentize:          Segmentize geometries of a vector dataset.
  - select:              Select a subset of fields from a vector dataset.
  - set-field-type:      Modify the type of a field of a vector dataset.
  - set-geom-type:       Modify the geometry type of a vector dataset.
  - simplify:            Simplify geometries of a vector dataset.
```

(continues on next page)

(continued from previous page)

```
  - simplify-coverage:   Simplify shared boundaries of a polygonal vector dataset.
  - sort:                Spatially order the features in a layer
  - sql:                 Apply SQL statement(s) to a dataset.
  - swap-xy:             Swap X and Y coordinates of geometries of a vector dataset.
  - update:              Update an existing vector dataset with an input vector dataset.

Common Options:
  -h, --help              Display help message and exit
  --json-usage            Display usage as JSON document and exit
  --config <KEY>=<VALUE>  Configuration option [may be repeated]

Options:
  --drivers               Display vector driver list as JSON document and exit
```

Available sub-commands

- *gdal vector clip*
- *gdal vector concat*
- *gdal vector convert*
- *gdal vector create*
- *gdal vector edit*
- *gdal vector filter*
- *gdal vector index*
- *gdal vector info*
- *gdal vector export-schema*
- *gdal vector layer-algebra*
- *gdal vector partition*
- *gdal vector pipeline*
- *gdal vector rasterize*
- *gdal vector rename-layer*
- *gdal vector reproject*
- *gdal vector select*
- *gdal vector set-field-type*
- *gdal vector set-geom-type*
- *gdal vector sql*
- *gdal vector update*

Examples

Example 1: Getting information on the file poly.gpkg (with text output)

```
$ gdal vector info poly.gpkg
```

Example 2: Converting file poly.gpkg to Esri File Geodatabase

```
$ gdal vector convert --format=OpenFileGDB poly.gpkg poly.gdb
```

Example 3: Check if the vector Parquet driver is installed

Bash

PowerShell

```
gdal vector --formats | grep Parquet
```

```
gdal vector --formats | Select-String "Parquet"
```

5.1.4.2 gdal vector buffer

Added in version 3.12.

Synopsis

```
Usage: gdal vector buffer [OPTIONS] <INPUT> <OUTPUT> <DISTANCE>

Compute a buffer around geometries of a vector dataset.

Positional arguments:
  -i, --input <INPUT>                                    Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]
  --distance <DISTANCE>                                  Distance to which to extend the␣
↪geometry. [required]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>               Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
```

(continues on next page)

(continued from previous page)

```
  --lco, --layer-creation-option <KEY>=<VALUE>         Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                          Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                             Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                    Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                             Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                       Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                        Output layer name [not available␣
↪in pipelines]
  --skip-errors                                        Skip errors when writing features␣
↪[not available in pipelines]
  --active-layer <ACTIVE-LAYER>                        Set active layer (if not␣
↪specified, all)
  --active-geometry <ACTIVE-GEOMETRY>                  Geometry field name to which to␣
↪restrict the processing (if not specified, all)
  --endcap-style <ENDCAP-STYLE>                        Endcap style.. ENDCAP-
↪STYLE=round|flat|square (default: round)
  --join-style <JOIN-STYLE>                            Join style.. JOIN-
↪STYLE=round|mitre|bevel (default: round)
  --mitre-limit <MITRE-LIMIT>                          Mitre ratio limit (only affects␣
↪mitered join style). (default: 5)
  --quadrant-segments <QUADRANT-SEGMENTS>              Number of line segments used to␣
↪approximate a quarter circle. (default: 8)
  --side <SIDE>                                        Sets whether the computed buffer␣
↪should be single-sided or not.. SIDE=both|left|right (default: both)

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                  Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                    Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>      Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                             Upsert features (implies 'append
↪') [not available in pipelines]
                                                       Mutually exclusive with --append
```

Description

gdal vector buffer computes a POLYGON or MULTIPOLYGON that represents all points whose distance from a geometry/geography is less than or equal to a given distance. A negative distance shrinks the geometry rather than expanding it. A negative distance may shrink a polygon completely, in which case POLYGON EMPTY is returned. For points and lines negative distances always return empty results.

See the PostGIS ST_Buffer docs for additional graphical illustrations of the effect of the different parameters.

The output dataset is always written as `MULTIPOLYGON`, even when the input contains only simple `POLYGON` geometries. Note that negative buffer distances may also produce `MULTIPOLYGON` results.

Using a buffer distance of 0 can sometimes repair invalid geometries, but the recommended approach is to use **gdal**

vector make-valid for reliable geometry validation and repair.

This command can also be used as a step of *gdal vector pipeline*.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

> **Note**
>
> This command requires a GDAL build against the GEOS library.

Program-Specific Options

--distance `<DISTANCE>`

Radius of the buffer around the input geometry. The unit of the distance is in georeferenced units of the source layer.

--endcap-style `round|flat|square`

Specifies the end cap style of the generated buffer. Default is `round`.

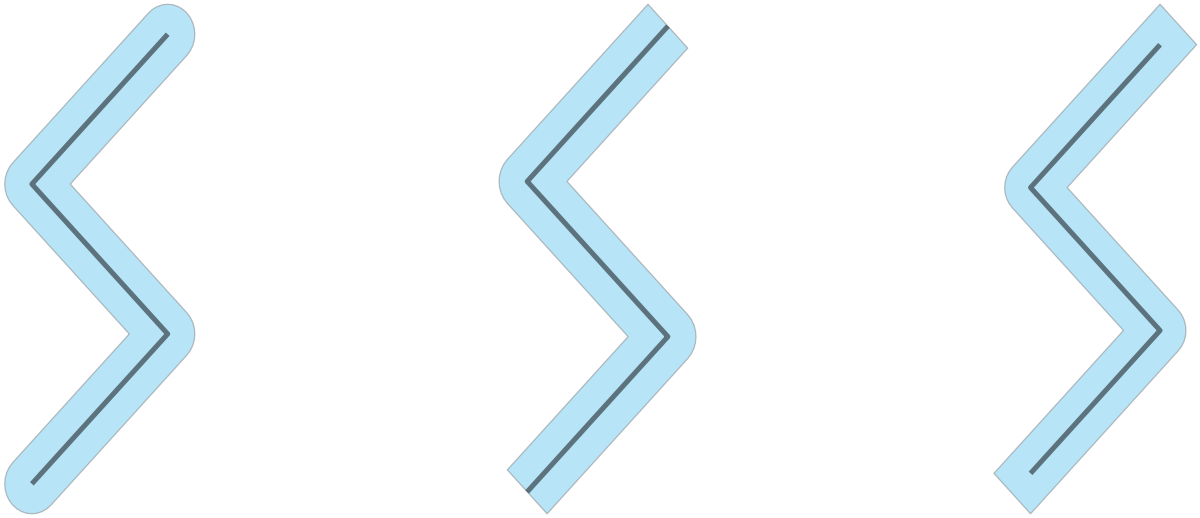


Fig. 5: Buffered lines with `endcap-style=round`, `endcap-style=flat` and `endcap-style=square`

--join-style `round|mitre|bevel`

Sets the join style for outside (reflex) corners between line segments. Default is `round`.

--input-layer `<NAME>`

Specifies one or more layer names to read and process. By default, all layers will be read and processed. To read and write all layers but only process a subset, use *`--active-layer`*.

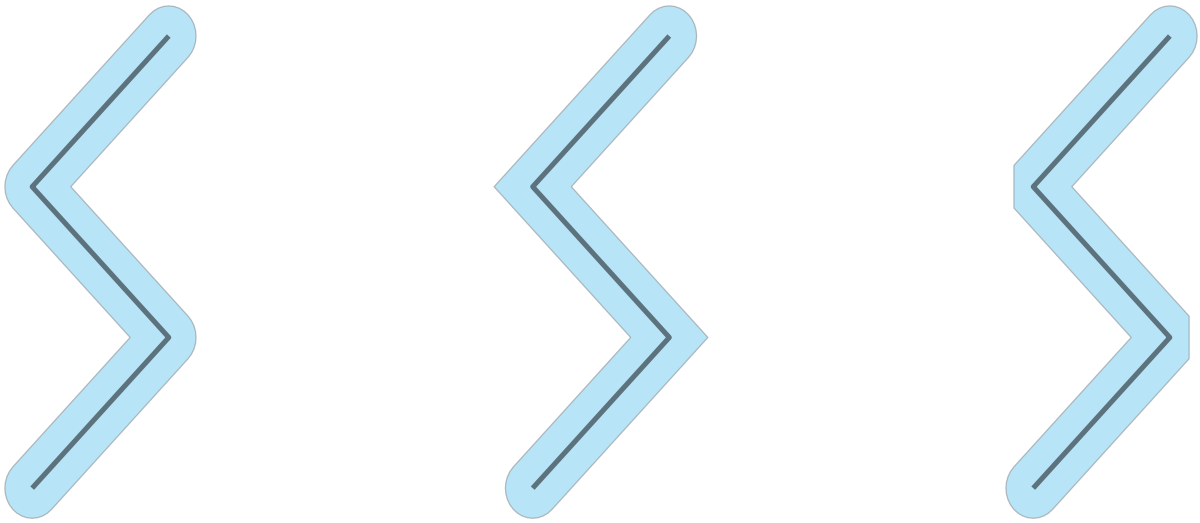


Fig. 6: Buffered lines with `join-style=round`, `join-style=mitre` and `join-style=bevel`

--mitre-limit <MITRE-LIMIT>

Sets the limit on the mitre ratio used for very sharp corners. This option only applies when `join-style=mitre` is used.

Default is 5.

The mitre ratio is the ratio of the distance from the corner to the end of the mitred offset corner. When two line segments meet at a sharp angle, a miter join will extend far beyond the original geometry. (and in the extreme case will be infinitely far.) To prevent unreasonable geometry, the mitre limit allows controlling the maximum length of the join corner. Corners with a ratio which exceed the limit will be beveled.

--output-layer <NAME>

Name of the layer to which output should be written. If not specified, the output layer will have the same name as the input layer.

--quadrant-segments <QUADRANT-SEGMENTS>

Sets the number of line segments used to approximate an angle fillet in round joins.

This determines the maximum error in the approximation to the true buffer curve. The default value of 8 gives less than 2% max error in the buffer distance. For a max error of < 1%, use QS = 12. For a max error of < 0.1%, use QS = 18. The error is always less than the buffer distance (in other words, the computed buffer curve is always inside the true curve).

--side `both|left|right`

Sets whether the computed buffer should be single-sided or on both sides (default).

`left` indicates that the buffer is generated on the left-hand side of the line, and `right` indicates that the buffer is generated on the right-hand side of the line, determined by the order of the line's vertices (i.e., the direction in which the line is drawn).

Single-side buffering is only applicable to LINESTRING geometry and does not affect POINT or POLYGON geometries, and the end cap style is forced to square.

Fig. 7: Lines with a `distance=2` buffer, and `mitre-limit=5` on the left, and `mitre-limit=1` on the right

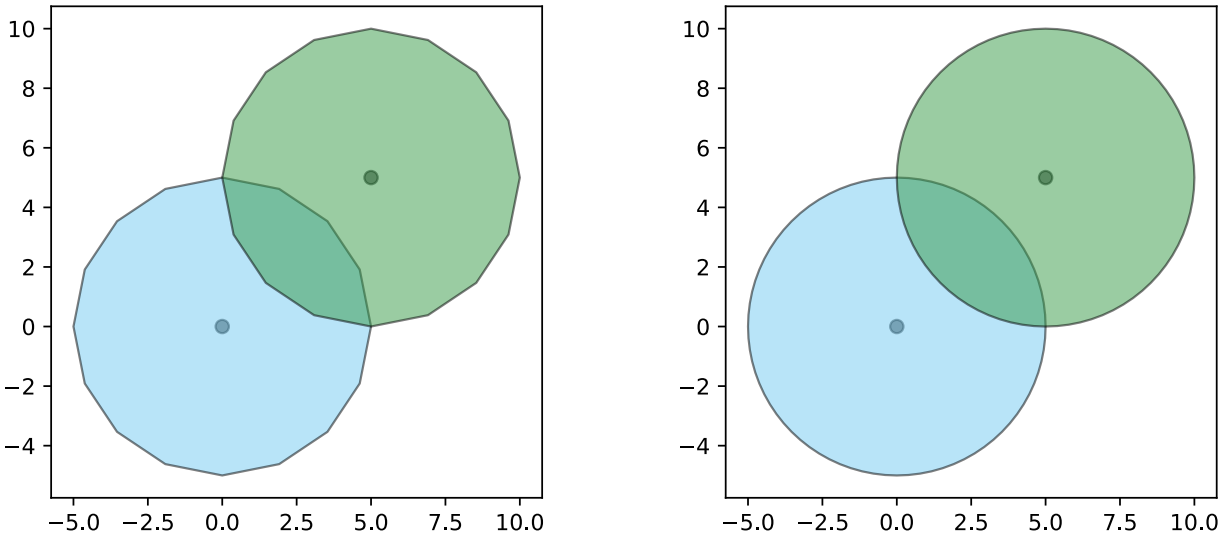


Fig. 8: Buffered points with `quadrant-segments=4` and `quadrant-segments=20`

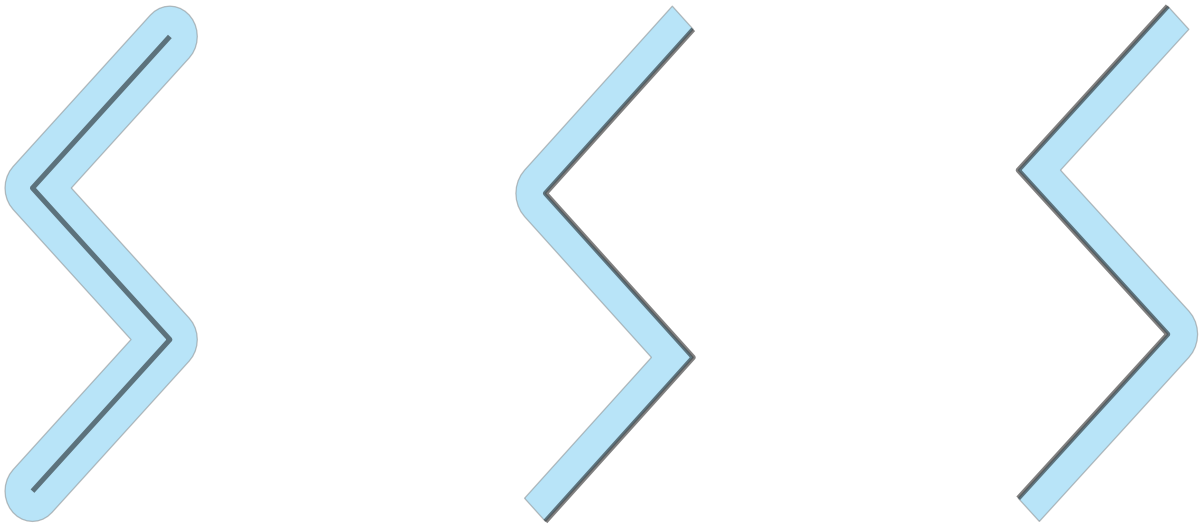


Fig. 9: Buffered lines with `side=both`, `side=left` and `side=right`

Standard Options

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--active-layer <ACTIVE-LAYER>

Set the active layer. When it is specified, only the layer specified by its name will be subject to the processing. Other layers will be not modified. If this option is not specified, all layers will be subject to the processing.

--active-geometry <ACTIVE-GEOMETRY>

Set the active geometry field from its name. When it is specified, only the specified geometry field will be subject to the processing. Other geometry fields will be not modified. If this option is not specified, all geometry fields will be subject to the processing. This option can be combined together with *--active-layer*.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Compute a buffer of one km around input geometries (assuming the CRS is in meters)

```
$ gdal vector buffer --distance=1000 in.gpkg out.gpkg --overwrite
```

Example 2: Buffer a point dataset with 20 segments per quadrant (instead of default 8) to better approximate a circle

```
$ gdal vector buffer --distance=20 --quadrant-segments=20 points.gpkg point_buffers.gpkg
```

Example 3: Buffer lines to the left using join and end cap styles

```
$ gdal vector buffer --distance=10 --endcap-style=flat --side=left --join-style=mitre --
↪mitre-limit=2 lines.gpkg lines-buffer-endcap.gpkg
```

Example 4: Clip and buffer a zipped line dataset as part of a pipeline

Bash

PowerShell

```
gdal vector pipeline \
! read "/vsizip/lines.zip" \
! clip --bbox 647903,6863599,648703,6864399 \
! buffer --distance=5 \
! write line-buffer.gpkg --overwrite
```

```
gdal vector pipeline `
! read "/vsizip/lines.zip" `
! clip --bbox 647903,6863599,648703,6864399 `
! buffer --distance=5 `
! write line-buffer.gpkg --overwrite
```

5.1.4.3 gdal vector check-coverage

Added in version 3.12.

Synopsis

```
Usage: gdal vector check-coverage [OPTIONS] <INPUT> <OUTPUT>
```

(continues on next page)

(continued from previous page)

```
Check a polygon coverage for validity

Positional arguments:
  -i, --input <INPUT>                                    Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>               Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>           Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                               Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                      Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                               Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                         Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                          Output layer name [not available␣
↪in pipelines]
  --skip-errors                                          Skip errors when writing features␣
↪[not available in pipelines]
  --no-create-empty-layers                               Avoid creating layers to which no␣
↪features will be written [not available in pipelines]
  --include-valid                                        Include valid inputs in output,␣
↪with empty geometry
  --geometry-field <GEOMETRY-FIELD>                      Name of geometry field to check
  --maximum-gap-width <MAXIMUM-GAP-WIDTH>                Maximum width of a gap to be␣
↪flagged

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                    Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                      Open options [may be repeated]␣
↪[not available in pipelines]
```

(continues on next page)

(continued from previous page)

```
  --output-oo, --output-open-option <KEY>=<VALUE>      Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                             Upsert features (implies 'append
↪') [not available in pipelines]
                                                       Mutually exclusive with --append
```

Description

gdal vector check-coverage checks whether a polygon dataset forms a valid coverage, meaning:

- no polygons overlap
- all shared edges have nodes at the same locations
- any gaps between polygons are larger than a specified width

For each polygon not satisfying these criteria, a linear feature with the geometry of the invalid edge is output. If the coverage is valid, the output dataset will be empty unless `--include-valid` is used.

It is assumed that the individual polygons are themselves valid according to the OGC Simple Features standard. This can be checked by *gdal vector check-geometry*.

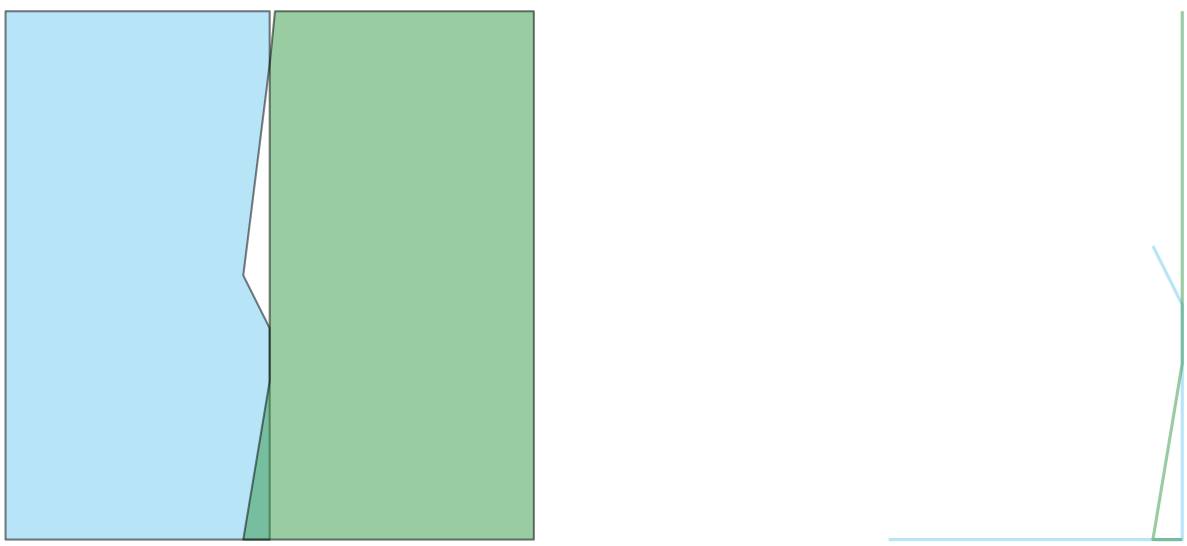


Fig. 10: Polygon dataset (left) and locations of invalid coverage edges (right).

> **Note**
>
> This command requires a GDAL build against the GEOS library (version 3.12 or greater).

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

> **Note**
>
> However this algorithm is not natively streaming compatible. Consequently a in-memory temporary dataset will be generated, which may cause significant processing time at opening.

Program-Specific Options

--geometry-field

Specify the name of the geometry field to test, for layers having multiple geometry fields. By default the first geometry field will be used.

--include-valid

Include features for valid geometries in the output, maintaining 1:1 correspondence between input and output features. The geometry of valid features will be an empty geometry.

--input-layer

Specifies the name of the layer to process. By default, all layers will be processed.

--output-layer

Specifies the name of the layer to which features will be written. If not specified and there is a single input layer, the name "invalid_edge" will be used. If not specified and there are multiple input layers, features for each input layer will be written to a separate output layer in the format "invalid_edge_{INPUT_LAYER}".

--maximum-gap-width <MAXIMUM-GAP-WIDTH>

Defines the largest area that should be considered a gap.

Standard Options

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the

layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--no-create-empty-layers

Added in version 3.13.

Avoid creating layers to which no features will be written.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of `--append` where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Output coverage errors in a TopoJSON file to a GeoJSON file

```
$ gdal vector check-coverage "https://cdn.jsdelivr.net/npm/us-atlas@3/counties-albers-
↪10m.json" --layer counties counties-errors.geojson
```

Example 2: Output invalid coverage features to the command-line as part of a pipeline

Bash

PowerShell

```
gdal vector pipeline \
    ! read "https://cdn.jsdelivr.net/npm/us-atlas@3/counties-albers-10m.json" --layer␣
↪counties \
    ! check-coverage \
    ! info --features
```

```
gdal vector pipeline `
    ! read "https://cdn.jsdelivr.net/npm/us-atlas@3/counties-albers-10m.json" --layer␣
↪counties `
    ! check-coverage `
    ! info --features
```

5.1.4.4 gdal vector check-geometry

Added in version 3.12.

Synopsis

```
Usage: gdal vector check-geometry [OPTIONS] <INPUT> <OUTPUT>

Check a dataset for invalid geometries

Positional arguments:
  -i, --input <INPUT>                                Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                              Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                         Display help message and exit
  --json-usage                                       Display usage as JSON document␣
```

(continues on next page)

(continued from previous page)

```
↪and exit
  --config <KEY>=<VALUE>                                Configuration option [may be␣
↪repeated]
  -q, --quiet                                           Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>              Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                 Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>          Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                           Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                              Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                     Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                              Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                        Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                         Output layer name [not available␣
↪in pipelines]
  --skip-errors                                         Skip errors when writing features␣
↪[not available in pipelines]
  --no-create-empty-layers                              Avoid creating layers to which no␣
↪features will be written [not available in pipelines]
  --include-field <INCLUDE-FIELD>                       Fields from input layer to␣
↪include in output (special values: ALL and NONE) (default: NONE) [may be repeated]
  --include-valid                                       Include valid inputs in output,␣
↪with empty geometry
  --geometry-field <GEOMETRY-FIELD>                     Name of geometry field to check

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                   Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                     Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>       Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                              Upsert features (implies 'append
↪') [not available in pipelines]
                                                        Mutually exclusive with --append
```

Description

gdal vector check-geometry checks that individual elements of a dataset are valid and simple according to the OGC Simple Features standard. For each invalid or non-simple feature, it will output a description and, in most cases, a point location of the error.

The following checks are performed, depending on the input geometry type:

- Polygons and MultiPolygons are checked for validity. A single point error point will be reported even if there are multiple causes of invalidity.
- LineStrings and MultiLineStrings are checked for simplicity. All self-intersection locations will be reported if GDAL is built using version 3.14 or later of the GEOS library. With earlier versions, self-intersection locations are not reported.
- GeometryCollections are checked that their individual elements are valid / simple. A single error point will be reported even if there are multiple causes of invalidity.
- Other geometry types are not checked.

Validity/simplicity checking is performed by the GEOS library and should be consistent with results of software such as PostGIS, QGIS, and shapely that also use that library. GEOS does not consider repeated points to be a cause of invalidity or non-simplicity. For more information about validity testing performed by **`gdal vector check-geometry`**, and examples of reported errors, see *Geometry Validity*.

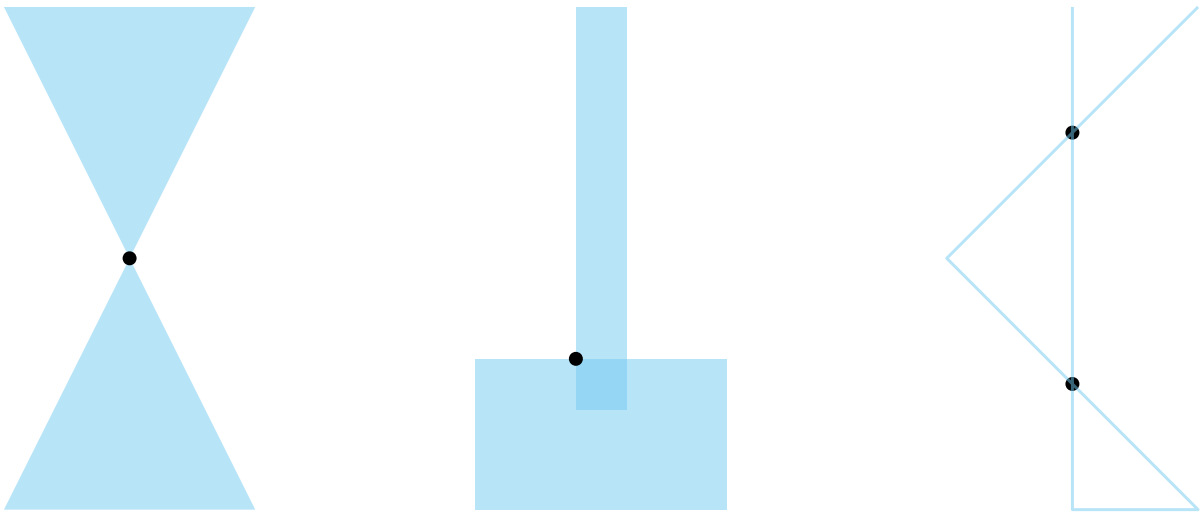


Fig. 11: Error locations reported by **`gdal vector check-geometry`** for Polygon, MultiPolyon, and LineString inputs.

Warning

Curved geometries are linearized before converting to GEOS. Linearized geometries may be valid/simple where the original geometries are not, and vice-versa.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Note

However this algorithm is not natively streaming compatible. Consequently a in-memory temporary dataset will be generated, which may cause significant processing time at opening.

Program-Specific Options

--geometry-field

Specify the name of the geometry field to test, for layers having multiple geometry fields. By default the first geometry field will be used.

--include-field

Added in version 3.12.1.

Optional field(s) to copy from the input features to the output. Since GDAL 3.13, the value `ALL` can be used to include all fields from the source layer.

--include-valid

Include features for valid/simple geometries in the output, maintaining 1:1 correspondence between input and output features.

--input-layer `<INPUT-LAYER>`

Specifies the name of one or more layers to process. By default, all layers will be processed.

--output-layer

Specifies the name of the layer to which features will be written. If not specified and there is a single input layer, the name "error_location" will be used. If not specified and there are multiple input layers, features for each input layer will be written to a separate output layer in the format "error_location_{INPUT_LAYER}".

Standard Options

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option `<NAME>=<VALUE>`

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format `<format>`

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--no-create-empty-layers

Added in version 3.13.

Avoid creating layers to which no features will be written.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Print invalidity locations to console

```
$ gdal vector check-geometry ne_10m_admin_0_countries.shp \
         --quiet \
         -f CSV \
         --lco GEOMETRY=AS_XY \
         --lco SEPARATOR=TAB \
         /vsistdout/
# X       Y       error
# 35.6210871060001       23.1392929140001        Ring Self-intersection
```

5.1.4.5 gdal vector clean-coverage

Added in version 3.12.

Synopsis

```
Usage: gdal vector clean-coverage [OPTIONS] <INPUT> <OUTPUT>

Alter polygon boundaries to make shared edges identical, removing gaps and overlaps

Positional arguments:
  -i, --input <INPUT>                                   Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                 Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                            Display help message and exit
  --json-usage                                          Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                Configuration option [may be␣
↪repeated]
  -q, --quiet                                           Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>              Input layer name(s) [may be␣
```

(continues on next page)

(continued from previous page)

```
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>         Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                          Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                             Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                    Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                             Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                       Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                        Output layer name [not available␣
↪in pipelines]
  --skip-errors                                        Skip errors when writing features␣
↪[not available in pipelines]
  --active-layer <ACTIVE-LAYER>                        Set active layer (if not␣
↪specified, all)
  --snapping-distance <SNAPPING-DISTANCE>              Distance tolerance for snapping␣
↪nodes
  --merge-strategy <MERGE-STRATEGY>                    Algorithm to assign overlaps to␣
↪neighboring polygons. MERGE-STRATEGY=longest-border|max-area|min-area|min-index
  --maximum-gap-width <MAXIMUM-GAP-WIDTH>              Maximum width of a gap to be␣
↪closed

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                  Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                    Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>      Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                             Upsert features (implies 'append
↪') [not available in pipelines]
                                                       Mutually exclusive with --append
```

Description

gdal vector clean-coverage modifies boundaries of a polygonal dataset, such that gaps and overlaps between features are removed and shared edges are defined using the same vertices. The resulting dataset will form a polygonal coverage that can be used with *gdal vector simplify-coverage*.

This command can also be used as a step of *gdal vector pipeline*, although it requires loading the entire dataset into memory at once.

> **Note**
>
> This command requires a GDAL build against the GEOS library (version 3.14 or greater).

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

> **Note**
>
> However this algorithm is not natively streaming compatible. Consequently a in-memory temporary dataset will be generated, which may cause significant processing time at opening.

Program-Specific Options

`--input-layer`

Specifies the name of the layer to process. By default, all layers will be processed.

`--maximum-gap-width` `<MAXIMUM-GAP-WIDTH>`

Defines the largest area that should be considered a "gap" and merged into an adjacent polygon. Gaps will be merged unless a circle with radius larger than the specified tolerance can be inscribed within the gap. The default maximum gap width is zero, meaning that gaps are not closed.

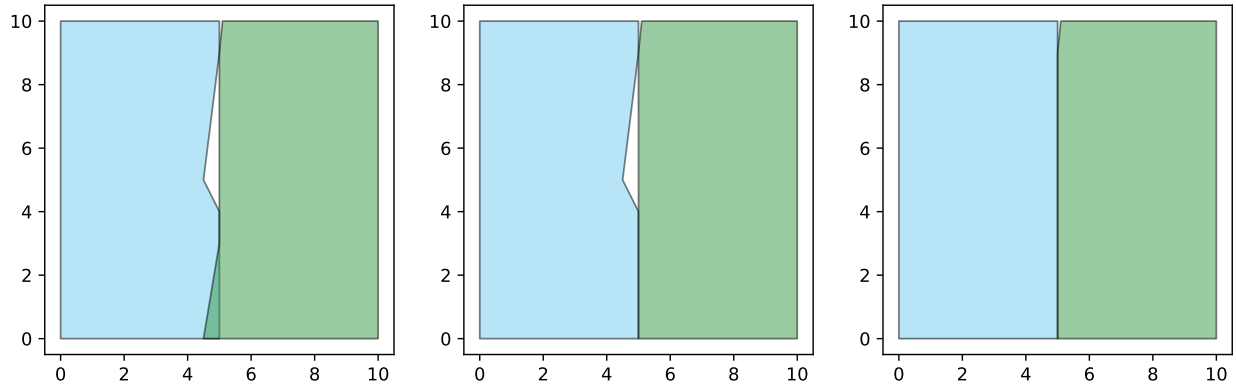


Fig. 12: Polygon dataset before cleaning (left), after cleaning with default parameters (center), and after cleaning with `--maximum-gap-width 1` (right).

`--merge-strategy` `<MERGE-STRATEGY>`

Method by which overlaps or gaps should be added to adjacent polygons. Options include: - longest-border (default): add areas to the polygon with which the longest border is shared - max-area: add areas to the largest adjacent polygon - min-area: add areas to the smallest adjacent polygon - min-index: add areas to the adjacent polygon that was read first

`--output-layer`

Specifies the name of the layer to which features will be written. By default, the names of the output layers will be the same as the names of the input layers.

`--snapping-distance` `<SNAPPING-DISTANCE>`

Controls the node snapping step, when nearby vertices are snapped together. By default, an automatic snapping distance is determined based on an analysis of the input. Set to zero to turn off all snapping.

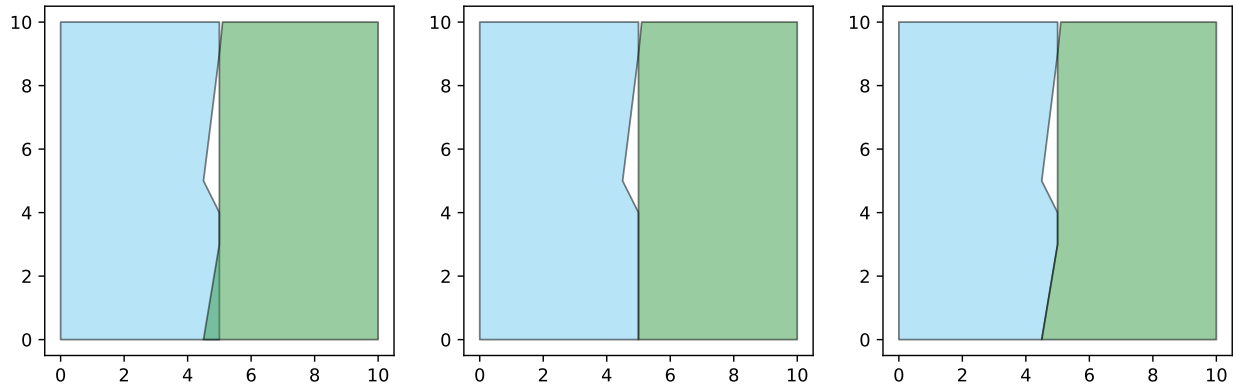


Fig. 13: Polygon dataset before cleaning (left), after cleaning with "longest-border" merge strategy (center) and `--merge-strategy max-area` (right).

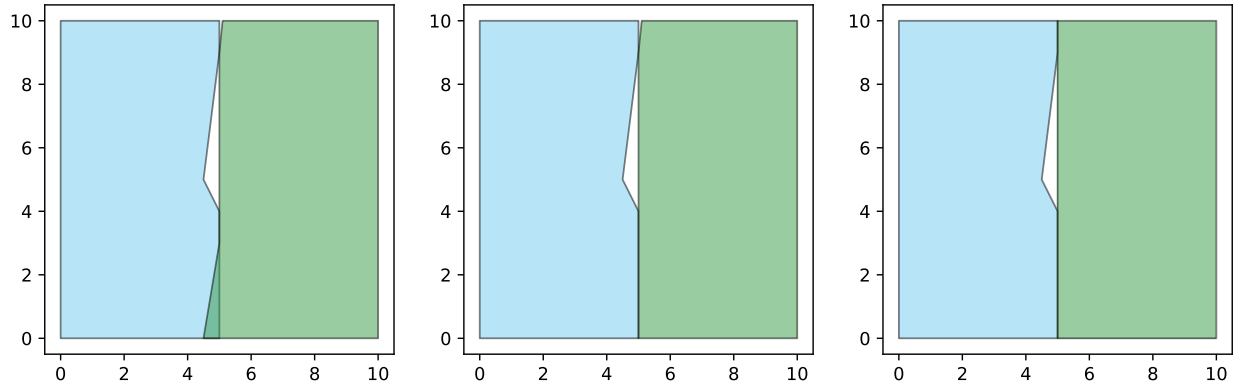


Fig. 14: Polygon dataset before cleaning (left), after cleaning with default snapping distance (center), and a more aggressive `--snapping-distance 0.2` (right). Note the movement in the upper-left corner of the polygon on the right.

Standard Options

--active-layer <ACTIVE-LAYER>

Set the active layer. When it is specified, only the layer specified by its name will be subject to the processing. Other layers will be not modified. If this option is not specified, all layers will be subject to the processing.

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Create and then simplify a polygonal coverage

```
$ gdal vector pipeline read ne_10m_admin_0_countries.shp ! \
                       make-valid ! \
                       clean-coverage ! \
                       simplify-coverage --tolerance 1 ! \
                       write countries.shp
```

5.1.4.6 gdal vector clip

Added in version 3.11.

Synopsis

```
Usage: gdal vector clip [OPTIONS] <INPUT> <OUTPUT>

Clip a vector dataset.

Positional arguments:
  -i, --input <INPUT>                                    Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>               Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>           Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                               Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                      Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                               Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                         Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                          Output layer name [not available␣
↪in pipelines]
  --skip-errors                                          Skip errors when writing features␣
↪[not available in pipelines]
  --active-layer <ACTIVE-LAYER>                          Set active layer (if not␣
↪specified, all)
  --bbox <BBOX>                                          Clipping bounding box as xmin,
↪ymin,xmax,ymax
                                                         Mutually exclusive with --
↪geometry, --like
  --bbox-crs <BBOX-CRS>                                  CRS of clipping bounding box
  --geometry <GEOMETRY>                                  Clipping geometry (WKT or GeoJSON)
                                                         Mutually exclusive with --bbox, --
↪like
```

(continues on next page)

(continued from previous page)

```
  --geometry-crs <GEOMETRY-CRS>                          CRS of clipping geometry
  --like <DATASET>                                       Dataset to use as a template for␣
↪bounds
                                                         Mutually exclusive with --bbox, --
↪geometry
  --like-sql <SELECT-STATEMENT>                          SELECT statement to run on the
↪'like' dataset
                                                         Mutually exclusive with --like-
↪where
  --like-layer <LAYER-NAME>                              Name of the layer of the 'like'␣
↪dataset
  --like-where <WHERE-EXPRESSION>                        WHERE SQL clause to run on the
↪'like' dataset
                                                         Mutually exclusive with --like-sql

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                    Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                      Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>        Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                               Upsert features (implies 'append
↪') [not available in pipelines]
                                                         Mutually exclusive with --append
```

Description

gdal vector clip can be used to clip a vector dataset using georeferenced coordinates.

Either *--bbox*, *--geometry* or *--like* must be specified.

`clip` can also be used as a step of *gdal vector pipeline*.

Clipping sometimes results in geometries that are of a type not compatible with the geometry type. This program splits multi-geometries or geometry collections into their parts for layers that are of a single geometry type (such as point, linestring, polygon) and only keeps the parts that are of that type. It also promotes single geometry types (e.g. polygons) to multi geometry types (e.g. multi-polygons). If the user needs to preserve any type of intersection, the layer must use the `wkbUnknown` (any geometry) type.

Program-Specific Options

--bbox `<xmin>,<ymin>,<xmax>,<ymax>`

Bounds to which to clip the dataset. They are assumed to be in the CRS of the input dataset, unless *--bbox-crs* is specified. The X and Y axis are the "GIS friendly ones", that is X is longitude or easting, and Y is latitude or northing. Mutually exclusive with *--like* and *--geometry*.

--bbox-crs `<CRS>`

CRS in which the <xmin>,<ymin>,<xmax>,<ymax> values of *--bbox* are expressed. If not specified, it is assumed to be the CRS of the input dataset. Note that specifying --bbox-crs does not involve doing vector reprojection. Instead, the bounds are reprojected from the bbox-crs to the CRS of the input dataset.

--geometry `<WKT_or_GeoJSON>`

Geometry as a WKT or GeoJSON string to which to clip the dataset. If the input geometry is GeoJSON, its CRS is

assumed to be WGS84, unless there is a CRS defined in the GeoJSON geometry or *--geometry-crs* is specified. If the input geometry is WKT, its CRS is assumed to be the one of the input dataset, unless *--geometry-crs* is specified. The X and Y axis are the "GIS friendly ones", that is X is longitude or easting, and Y is latitude or northing. Mutually exclusive with *--bbox* and *--like*.

--geometry-crs <CRS>

CRS in which the coordinates values of *--geometry* are expressed. If not specified, it is assumed to be the CRS of the input dataset. Note that specifying --geometry-crs does not involve doing vector reprojection. Instead, the bounds are reprojected from the geometry-crs to the CRS of the input dataset.

--like <DATASET>

Vector or raster dataset to use as a template for bounds. If the specified dataset is a raster, its rectangular bounds are used as the clipping geometry. If the specified dataset is a vector dataset, its polygonal geometries are unioned together to form the clipping geometry (beware that the result union might not be perfect if there are gaps between individual polygon features). If several layers are present, *--like-sql* or *--like-layer* must be specified. Mutually exclusive with *--bbox* and *--geometry*.

--like-layer <LAYER-NAME>

Select the named layer from the vector clip dataset. Mutually exclusive with *--like-sql*

--like-sql <SELECT-STATEMENT>

Select desired geometries from the vector clip dataset using an SQL query. e.g `SELECT geom FROM my_layer WHERE country = 'France'`. Mutually exclusive with *--like-layer* and *--like-where*

--like-where <WHERE-EXPRESSION>

Restrict desired geometries from vector clip dataset layer based on an attribute query. e.g `country = 'France'`.

Standard Options

--active-layer <ACTIVE-LAYER>

Set the active layer. When it is specified, only the layer specified by its name will be subject to the processing. Other layers will be not modified. If this option is not specified, all layers will be subject to the processing.

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--input-layer <INPUT-LAYER>

Specifies the name of one or more layers to process. By default, all layers will be processed. To read and write all layers but only process a subset, use *--active-layer*.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-layer <NAME>

Specifies the name of the layer to which features will be written. If not specified, output layer names will be the same as input layer names.

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Clip a GeoPackage file to the bounding box from longitude 2, latitude 49, to longitude 3, latitude 50 in WGS 84

```
$ gdal vector clip --bbox=2,49,3,50 --bbox-crs=EPSG:4326 in.gpkg out.gpkg --overwrite
```

5.1.4.7 gdal vector combine

Added in version 3.13.

Synopsis

```
Usage: gdal vector combine [OPTIONS] <INPUT> <OUTPUT>

Combine features into collections

Positional arguments:
  -i, --input <INPUT>                                    Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>               Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
```

(continues on next page)

(continued from previous page)

```
  --co, --creation-option <KEY>=<VALUE>                Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>         Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                          Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                             Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                    Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                             Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                       Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                        Output layer name [not available␣
↪in pipelines]
  --skip-errors                                        Skip errors when writing features␣
↪[not available in pipelines]
  --group-by <GROUP-BY>                                Names of field(s) by which inputs␣
↪should be grouped [may be repeated]
  --keep-nested                                        Avoid combining the components of␣
↪multipart geometries
  --add-extra-fields <ADD-EXTRA-FIELDS>                Whether to add extra fields,␣
↪depending on if they have identical values within each group. ADD-EXTRA-
↪FIELDS=no|sometimes-identical|always-identical (default: no)

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                  Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                    Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>      Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                             Upsert features (implies 'append
↪') [not available in pipelines]
                                                       Mutually exclusive with --append
```

Description

gdal vector combine combines geometries into geometry collections.

The *--group-by* argument can be used to determine which features are combined.

By default, the parts of multipart geometries will be combined into an un-nested collection of the most specific type possible (e.g., `MultiPolygon` rather than `GeometryCollection`). For example, a Polygon and a two-part MultiPolygon would be combined into a three-part MultiPoygon. This is done because many GIS file formats and software packages do not handle nested GeometryCollections types. If the nested representation in the manner of PostGIS' `ST_Collect` is preferred (a two-component GeometryCollection containing the Polygon and MultiPolygon), then *--keep-nested* can be used.

`combine` can be used as a step of *gdal vector pipeline*.

Program-Specific Options

--group-by <FIELD_NAMES>

The names of fields whose unique values will be used to collect input geometries. Any fields not listed in *--group-by* will be removed from the source layer. If *--group-by* is omitted, the entire layer will be combined into a single feature.

--keep-nested

If input geometries have multiple parts, combine them into a GeometryCollection of the input geometries.

--add-extra-fields no|sometimes-identical|always-identical

Whether fields from source features that have the same value among the features belonging to a same group are copied to the corresponding output feature.

By default, no, only fields listed in *--group-by* are copied to the output layer.

When specifying sometimes-identical, fields, for which all input features within at least one output group have the same value, will be copied to the output layer schema. The value of the field is the value that is common for all input features of a group. If the input features within some group are not identical, the field value in the output for this group will be set to null.

When specifying always-identical, fields for which all input features within a given output group have the same value are copied to the output layer, provided this condition holds for all output groups.

For example, let's suppose we have four input features with the following content, and we group them by country

Table 2: Input features

name	country	country_fr	type
Mainland France	France	France	continental
Corsica	France	France	island
Mainland USA	USA	Etats-Unis d'Amérique	continental
Alaska	USA	Etats-Unis d'Amérique	continental

With sometimes-identical, the output will be:

Table 3: Output with sometimes-identical

country	country_fr	type
France	France	
USA	Etats-Unis d'Amérique	continental

The name field has been removed because it is distinct for each input feature. The type field appears as an output field, because at least for the USA, the two grouped input features that makes it have the same value. For France, its content is set to null, because its value is different in the two input features.

With always-identical, the output will be:

Table 4: Output with always-identical

country	country_fr
France	France
USA	Etats-Unis d'Amérique

The type field has been removed because its value is different in the two input features of France.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Standard Options

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

5.1.4.8 gdal vector concat

Added in version 3.11.

Synopsis

```
Usage: gdal vector concat [OPTIONS] <INPUTS>... <OUTPUT>

Concatenate vector datasets.

Positional arguments:
  -i, --input <INPUTS>                                      Input vector datasets [1..␣
↪values] [required] [not available in pipelines]
  -o, --output <OUTPUT>                                     Output vector dataset␣
↪[required] [not available in pipelines]

Common Options:
  -h, --help                                                Display help message and␣
↪exit
  --json-usage                                              Display usage as JSON␣
↪document and exit
  --config <KEY>=<VALUE>                                    Configuration option [may␣
↪be repeated]
```

(continues on next page)

(continued from previous page)

```
  -q, --quiet                                                   Quiet mode (no progress bar␣
↪or warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>                      Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>           Output format ("GDALG"␣
↪allowed) [not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                         Creation option [may be␣
↪repeated] [not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>                  Layer creation option [may␣
↪be repeated] [not available in pipelines]
  --overwrite                                                   Whether overwriting␣
↪existing output dataset is allowed [not available in pipelines]
  --update                                                      Whether to open existing␣
↪dataset in update mode [not available in pipelines]
  --overwrite-layer                                             Whether overwriting␣
↪existing output layer is allowed [not available in pipelines]
  --append                                                      Whether appending to␣
↪existing layer is allowed [not available in pipelines]
                                                                Mutually exclusive with --
↪upsert
  --skip-errors                                                 Skip errors when writing␣
↪features [not available in pipelines]
  --mode <MODE>                                                 Determine the strategy to␣
↪create output layers from source layers . MODE=merge-per-layer-name|stack|single␣
↪(default: merge-per-layer-name)
  --output-layer <OUTPUT-LAYER>                                 Name of the output vector␣
↪layer (single mode), or template to name the output vector layers (stack mode)
  --source-layer-field-name <SOURCE-LAYER-FIELD-NAME>           Name of the new field to␣
↪add to contain identification of the source layer, with value determined from 'source-
↪layer-field-content'
  --source-layer-field-content <SOURCE-LAYER-FIELD-CONTENT>  A string, possibly using
↪{AUTO_NAME}, {DS_NAME}, {DS_BASENAME}, {DS_INDEX}, {LAYER_NAME}, {LAYER_INDEX}
  --field-strategy <FIELD-STRATEGY>                             How to determine target␣
↪fields from source fields. FIELD-STRATEGY=union|intersection (default: union)
  -s, --input-crs <INPUT-CRS>                                   Input CRS
  -d, --output-crs <OUTPUT-CRS>                                 Output CRS

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                           Input formats [may be␣
↪repeated] [not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                             Open options [may be␣
↪repeated] [not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>               Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                                      Upsert features (implies
↪'append') [not available in pipelines]
                                                                Mutually exclusive with --
↪append
```

Description

gdal vector concat concatenates several source datasets.

It has 3 main modes:

- *--mode* = `merge-per-layer-name` (the default). The output dataset generated by the command will contain as many layers as there are different layer names in the source datasets. For example if there are 2 datasets, one with layers *a* and *b*, and the other one with layers *b* and *c*, 3 output layers will be created: *a*, *b* (merging the 2 source layers) and *c*.
- *--mode* = `stack`. The output dataset generated by the command will contain as many layers as there are layers in the source datasets. For example if there are 2 datasets `ds1` (with layers *a* and *b*) and `ds2` (with layers *b* and *c*), 4 output layers will be created: `ds1_a`, `ds1_b`, `ds2_b` and `ds2_c`.
- *--mode* = `single`. The output dataset generated by the command will contain one single layer, merging all layers in the source datasets.

When an output layer merges several source layer, by default the resulting schema will contain the union of all source fields. It is possible to select only the intersection with the *--field-strategy* set to `intersection`. Regarding the resulting CRS, by default the CRS of the source layer will be used as the target CRS, and features of other source layers that do no match this CRS will be reprojected to it. *--output-crs* can be used to select a given destination CRS.

This command can also be used as the first step of *gdal vector pipeline*.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

--field-strategy `union|intersection`

Determines how the schema of the target layer is built from the schemas of the input layers:

- `union` (default) to use a super-set of all the fields from all source layers.
- `intersection` to use a sub-set of all the common fields from all source layers.

--input-crs, -s `<INPUT-CRS>`

Set (override) input spatial reference. If not specified the SRS found in the input dataset will be used.

The coordinate reference systems that can be passed are anything supported by the OGRSpatialReference.SetFromUserInput() call, which includes EPSG Projected, Geographic or Compound CRS (i.e. EPSG:4296), a well known text (WKT) CRS definition, PROJ.4 declarations, or the name of a .prj file containing a WKT CRS definition.

If the SRS has an explicit vertical datum that points to a geoid grid, and the input dataset is a single band dataset, a vertical correction will be applied to the values of the dataset.

--mode `merge-per-layer-name|stack|single`

Determine the strategy to create output layers from source layers. See introductory paragraph for more details.

--output-crs, -d `<OUTPUT-CRS>`

Set output spatial reference. Inputs will be reprojected to this CRS if necessary.

The coordinate reference systems that can be passed are anything supported by the OGRSpatialReference.SetFromUserInput() call, which includes EPSG Projected, Geographic or Compound CRS (i.e.

EPSG:4296), a well known text (WKT) CRS definition, PROJ.4 declarations, or the name of a .prj file containing a WKT CRS definition.

If the SRS has an explicit vertical datum that points to a geoid grid, and the input dataset is a single band dataset, a vertical correction will be applied to the values of the dataset.

--output-layer <OUTPUT-LAYER>

Name of the output vector layer (in `single` mode, and the default is "merged"), or template to name the output vector layers in `stack` mode (the default value is `{AUTO_NAME}`). Not allowed in `merge-per-layer-name` mode.

The template in `stack` mode can be a string with the following variables that will be substituted with a value computed from the input layer being processed:

- `{AUTO_NAME}`: equivalent to `{DS_BASENAME}_{LAYER_NAME}` if both values are different, or `{LAYER_NAME}` when they are identical (case of shapefile).
- `{DS_NAME}`: name of the source dataset
- `{DS_BASENAME}`: base name of the source dataset
- `{DS_INDEX}`: index of the source dataset
- `{LAYER_NAME}`: name of the source layer
- `{LAYER_INDEX}`: index of the source layer

--source-layer-field-content <SOURCE-LAYER-FIELD-CONTENT>

If specified, the schema of the target layer will be extended with a new field (whose name is given by *--source-layer-field-name*, or `source_ds_lyr` otherwise), whose content is determined by the specified template (see *--output-layer* for variables that can be used).

--source-layer-field-name <SOURCE-LAYER-FIELD-NAME>

If specified, the schema of the target layer will be extended with a field whose name is the value of this option and whose content is determined *--source-layer-field-content*.

Standard Options

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--input-layer <INPUT-LAYER>

Specifies the name of one or more layers to process. By default, all layers will be processed.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Creating a GeoPackage stacking all input shapefiles in separate layers.

```
gdal vector concat --mode=stack *.shp out.gpkg
```

Example 2: Adding a field to indicate the source layer, and reprojecting to a single CRS

Concatenate the content of `france.shp` and `germany.shp` in `merged.shp`, reprojecting them to ETRS89, and add a 'country' field to each feature whose value is 'france' or 'germany' depending where it comes from:

```
gdal vector concat --mode=single --source-layer-field-name=country --output-
↪crs=EPSG:4258 france.shp germany.shp merged.shp
```

5.1.4.9 gdal vector concave-hull

Added in version 3.13.

Synopsis

```
Usage: gdal vector concave-hull [OPTIONS] <INPUT> <OUTPUT>

Compute the concave hull of geometries of a vector dataset.

Positional arguments:
  -i, --input <INPUT>                                    Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
```

(continues on next page)

(continued from previous page)

```
  -l, --layer, --input-layer <INPUT-LAYER>             Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>         Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                          Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                             Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                    Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                             Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                       Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                        Output layer name [not available␣
↪in pipelines]
  --skip-errors                                        Skip errors when writing features␣
↪[not available in pipelines]
  --active-layer <ACTIVE-LAYER>                        Set active layer (if not␣
↪specified, all)
  --active-geometry <ACTIVE-GEOMETRY>                  Geometry field name to which to␣
↪restrict the processing (if not specified, all)
  --ratio <RATIO>                                      Ratio controlling the concavity␣
↪[required]
  --allow-holes                                        Allow holes in the output polygon
  --tight                                              Whether the hull must follow the␣
↪outer boundaries of the input polygons

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                  Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                    Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>      Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                             Upsert features (implies 'append
↪') [not available in pipelines]
                                                       Mutually exclusive with --append
```

Description

gdal vector concave-hull computes the smallest concave polygon that covers the input geometry. Generally the result will be a Polygon, but for degenerate inputs it may be a LineString or a Point. It uses the *OGRGeometry::ConcaveHullOfPolygons()* method for polygonal input, and *OGRGeometry::ConcaveHull()* otherwise.

This command can also be used as a step of *gdal vector pipeline*.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

> **Note**
>
> This command requires a GDAL build against the GEOS library.

Program-Specific Options

--allow-holes

Allow result polygons to contain holes.

--tight

Whether the hull must follow the outer boundaries of the input polygons.

--ratio <RATIO>

Ratio controlling the concavity of the geometry.

A value of 1 produces the convex hull; decreasing values of the ratio generally produce polygons of increasing concavity (and with polygonal input, a value of 0 produces the original polygons).

Standard Options

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--active-layer <ACTIVE-LAYER>

Set the active layer. When it is specified, only the layer specified by its name will be subject to the processing. Other layers will be not modified. If this option is not specified, all layers will be subject to the processing.

--active-geometry <ACTIVE-GEOMETRY>

Set the active geometry field from its name. When it is specified, only the specified geometry field will be subject to the processing. Other geometry fields will be not modified. If this option is not specified, all geometry fields will be subject to the processing. This option can be combined together with `--active-layer`.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be

repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--input-layer <INPUT-LAYER>

Specifies the name of one or more layers to process. By default, all layers will be processed. To read and write all layers but only process a subset, use *--active-layer*.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-layer <NAME>

Specifies the name of the layer to which features will be written. If not specified, output layer names will be the same as input layer names.

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Examples

Example 1: Compute the concave hull of city locations in each country

Here the Natural Earth Populated Places dataset is used. To ensure that a polygonal hull is generated for each country, the input points are first buffered by a small amount. Otherwise, countries having only one or two cities would end up with a point or line geometry.

```
gdal vector pipeline ! \
    read ne_10m_populated_places.shp ! \
    buffer 0.1 ! \
    combine --group-by ADM0_A3 ! \
    concave-hull --ratio 0.5 ! \
    write country_city_hulls.gpkg
```

Example 2: Compute the concave hull of polygons of all countries, starting with their internal administrative divisions.

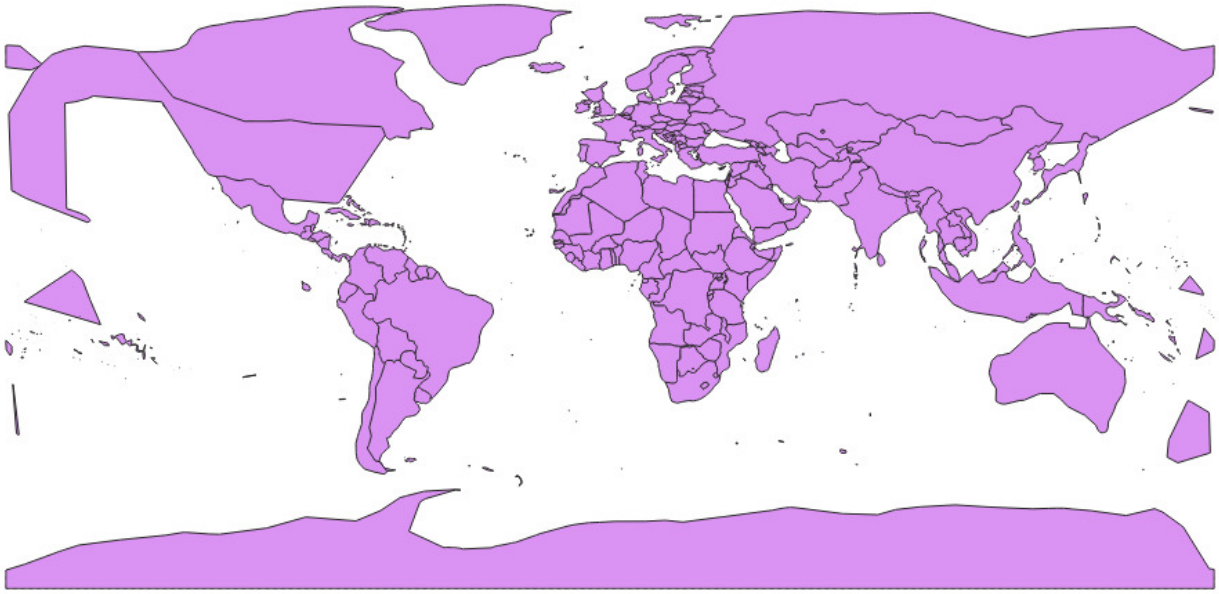
We use the "Admin 1 - States, Provinces" dataset, so that islands and oversea territories are in distinct polygonal features, to get a better looking result than if we started from "Admin 0 - Countries", where for example Bouvet Island in the South Atlantic is in the same multipolygon feature as mainland Norway.

The *combine* stage helps merging together administrative divisions of a same country with geographic proximity.

The *dissolve* stage transforms each geometry collection resulting from the previous stage into a valid polygon/multipolygon, as required by the concave hull by polygons algorithm.

The `sql` stage helps rendering smaller countries on top of larger ones.

```
gdal vector pipeline ! \
    read /vsizip//vsicurl/https://naciscdn.org/naturalearth/10m/cultural/ne_10m_admin_1_
↪states_provinces.zip ! \
    combine --group-by admin,geonunit ! \
    dissolve ! \
    concave-hull --ratio 0.1 ! \
    rename-layer --output-layer countries_concave_hull_of_polygons ! \
    sql --sql "SELECT * FROM countries_concave_hull_of_polygons ORDER BY ST_
↪Area(geometry) DESC" --dialect SQLite ! \
    write countries_concave_hull_of_polygons.gpkg
```



5.1.4.10 gdal vector convert

Added in version 3.11.

Synopsis

```
Usage: gdal vector convert [OPTIONS] <INPUT> <OUTPUT>

Convert a vector dataset.

Positional arguments:
  -i, --input <INPUT>                                    Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>               Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
```

(continues on next page)

(continued from previous page)

```
  --lco, --layer-creation-option <KEY>=<VALUE>         Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                          Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                             Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                    Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                             Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                       Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                        Output layer name [not available␣
↪in pipelines]
  --skip-errors                                        Skip errors when writing features␣
↪[not available in pipelines]
  --no-create-empty-layers                             Avoid creating layers to which no␣
↪features will be written [not available in pipelines]

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                  Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                    Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>      Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                             Upsert features (implies 'append
↪') [not available in pipelines]
                                                       Mutually exclusive with --append
```

Description

gdal vector convert can be used to convert data data between different formats.

Standard Options

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it

can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--input-layer <INPUT-LAYER>

Specifies the name of one or more layers to process. By default, all layers will be processed.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--no-create-empty-layers

Added in version 3.13.

Avoid creating layers to which no features will be written.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-layer <NAME>

Specifies the name of the layer to which features will be written. If not specified, output layer names will be the same as input layer names.

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

-q, --quiet

Suppress progress bar and some warning messages.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Converting file poly.shp to a GeoPackage

```
$ gdal vector convert poly.shp output.gpkg
```

Example 2: Add new layer from file line.shp to an existing GeoPackage, and rename it "lines"

```
$ gdal vector convert --update --output-layer=lines line.shp output.gpkg
```

Example 3: Append features from from file poly2.shp to an existing layer poly of a GeoPackage, without a progress bar

```
$ gdal vector convert --quiet --append --output-layer=poly poly2.shp output.gpkg
```

5.1.4.11 gdal vector convex-hull

Added in version 3.13.

Synopsis

```
Usage: gdal vector convex-hull [OPTIONS] <INPUT> <OUTPUT>

Compute the convex hull of geometries of a vector dataset.
```

(continues on next page)

(continued from previous page)

```
Positional arguments:
  -i, --input <INPUT>                                   Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                 Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                            Display help message and exit
  --json-usage                                          Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                Configuration option [may be␣
↪repeated]
  -q, --quiet                                           Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>              Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                 Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>          Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                           Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                              Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                     Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                              Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                        Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                         Output layer name [not available␣
↪in pipelines]
  --skip-errors                                         Skip errors when writing features␣
↪[not available in pipelines]
  --active-layer <ACTIVE-LAYER>                         Set active layer (if not␣
↪specified, all)
  --active-geometry <ACTIVE-GEOMETRY>                   Geometry field name to which to␣
↪restrict the processing (if not specified, all)

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                   Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                     Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>       Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                              Upsert features (implies 'append
↪') [not available in pipelines]
```

(continues on next page)

(continued from previous page)

```
                                                    Mutually exclusive with --append
```

Description

gdal vector convex-hull computes the smallest convex polygon that covers the input geometry. Generally the result will be a Polygon, but for degenerate inputs it may be a LineString or a Point.

This command can also be used as a step of *gdal vector pipeline*.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

> **Note**
>
> This command requires a GDAL build against the GEOS library.

Standard Options

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--active-layer <ACTIVE-LAYER>

Set the active layer. When it is specified, only the layer specified by its name will be subject to the processing. Other layers will be not modified. If this option is not specified, all layers will be subject to the processing.

--active-geometry <ACTIVE-GEOMETRY>

Set the active geometry field from its name. When it is specified, only the specified geometry field will be subject to the processing. Other geometry fields will be not modified. If this option is not specified, all geometry fields will be subject to the processing. This option can be combined together with `--active-layer`.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--input-layer <INPUT-LAYER>

Specifies the name of one or more layers to process. By default, all layers will be processed. To read and write all layers but only process a subset, use *--active-layer*.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-layer <NAME>

Specifies the name of the layer to which features will be written. If not specified, output layer names will be the same as input layer names.

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Examples

Example 1: Compute the convex hull of each country

```
gdal vector convex-hull countries.gpkg countries_convexhull.gpkg
```

Example 2: Compute the convex hull of city locations in each country

Here the Natural Earth Populated Places dataset is used. To ensure that a polygonal hull is generated for each country, the input points are first buffered by a small amount. Otherwise, countries having only one or two cities would end up with a point or line geometry.

```
gdal vector pipeline ! \
    read ne_10m_populated_places.shp ! \
    buffer 0.1 ! \
    combine --group-by ADM0_A3 ! \
    convex-hull ! \
    write country_city_hulls.gpkg
```

5.1.4.12 gdal vector create

Added in version 3.13.

Synopsis

```
Usage: gdal vector create [OPTIONS] <OUTPUT>

Create a vector dataset.

Positional arguments:
  -o, --output <OUTPUT>                                Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                           Display help message and exit
  --json-usage                                         Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                               Configuration option [may be␣
↪repeated]

Options:
```

(continues on next page)

(continued from previous page)

```
  -i, --like, --input <TEMPLATE-DATASET>                Input vector datasets [not␣
↪available in pipelines]
                                                        Mutually exclusive with --schema,␣
↪--field
  -l, --layer, --input-layer <INPUT-LAYER>              Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>   Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                 Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>          Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                           Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                              Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                     Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --output-layer <OUTPUT-LAYER>                         Output layer name [not available␣
↪in pipelines]
  --geometry-type <GEOMETRY-TYPE>                       Layer geometry type
  --geometry-field <GEOMETRY-FIELD>                     Name of the geometry field to␣
↪create (if supported by the output format) (default: geom)
  --crs <CRS>                                           Set CRS
  --fid <FID>                                           FID column name
  --schema <SCHEMA_JSON>                                Read OGR_SCHEMA and populate␣
↪field definitions from it
                                                        Mutually exclusive with --input, -
↪-field
  --field <NAME>:<TYPE>[(,<WIDTH>[,<PRECISION>])]>      Add a field definition to the␣
↪output layer [may be repeated]
                                                        Mutually exclusive with --input, -
↪-schema

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                   Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                     Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>       Output open options [may be␣
↪repeated] [not available in pipelines]
```

Description

gdal vector create creates one or more vector layers, creating the dataset if it doesn't exist or adding layers to an existing dataset if the underlying format supports it.

The layer fields, geometry type and coordinate reference system can be defined either by providing an existing dataset or an OGR_SCHEMA file/JSON as a template, or by explicitly specifying the various components.

gdal vector create can be used as the first step of a pipeline.

The following options are available:

Program-Specific Options

--output-layer <OUTPUT-LAYER>

Output layer name.

Only meaningful when creating a single layer dataset. It allows defining or overriding the name of the created layer.

-i, --like, --input <TEMPLATE-DATASET>

Vector dataset to use as a template for the created dataset.

If the template dataset contains multiple layers and the destination format does not support multiple layers the program will exit with an error unless *--input-layer* is used to specify which single layer from the input dataset should be used as a template.

Mutually exclusive with *--schema* and with *--field*, *--geometry-type*, *--geometry-field* and *--crs*.

--layer, --input-layer <INPUT-LAYER>

Layer(s) of the input dataset to use as a template for the created layer.

This option can be repeated multiple times to specify multiple layers.

This option is only meaningful when *--like* or *--schema* are used with an input dataset containing multiple layers.

--schema <OGR_SCHEMA>

File or JSON string containing the OGR_SCHEMA of the layer to create.

This can be a filename, a URL or JSON string conformant with the ogr_fields_override.schema.json schema

If the OGR_SCHEMA contains multiple layers and the destination format does not support multiple layers the program will exit with an error unless *--input-layer* is used to specify which single layer from the OGR_SCHEMA should be selected.

Mutually exclusive with *--like* and with *--field*, *--geometry-type*, *--geometry-field* and *--crs*.

--crs <CRS>

Defines the coordinate reference system of the created layer.

The coordinate reference systems that can be passed are anything supported by the *OGRSpatialReference::SetFromUserInput()* call, which includes EPSG Projected, Geographic or Compound CRS (i.e. EPSG:4296), a well known text (WKT) CRS definition, PROJ.4 declarations, or the name of a .prj file containing a WKT CRS definition.

Mutually exclusive with *--schema* and with *--like*.

--geometry-field <GEOMETRY-FIELD>

Name of the geometry field to create (if supported by the output format) (default: geom).

--geometry-type <GEOMETRY-TYPE>

Defines the geometry type of the created layer.

Layer geometry type to be one of `GEOMETRY`, `POINT`, `LINESTRING`, `POLYGON`, `MULTIPOINT`, `MULTILINESTRING`, `MULTIPOLYGON`, `GEOMETRYCOLLECTION`, `CURVE`, `CIRCULARSTRING`, `COMPOUNDCURVE`, `SURFACE`, `CURVEPOLYGON`, `MULTICURVE`, `MULTISURFACE`, `POLYHEDRALSURFACE` or `TIN`. `Z`, `M` or `ZM` suffixes can be appended to the above values to indicate the dimensionality.

Mutually exclusive with *--schema* and with *--like*.

--field <NAME>:<TYPE>[(,<WIDTH>[,<PRECISION>])]

Defines a field to be created in the layer.

This option can be repeated multiple times to create multiple fields.

The field type can be one of INTEGER, INTEGER64, REAL, STRING, DATE, TIME, DATETIME, BINARY, INTEGERLIST, REALLIST, STRINGLIST, INTEGER64LIST.

The width and precision can be specified for INTEGER, INTEGER64, REAL and STRING field types. For example, --field population:INTEGER(10,5) or --field name:STRING(255)

Mutually exclusive with *--schema* and with *--like*.

--fid <FID>

Defines the name of the Feature Identifier (FID) column, for drivers that support setting it (those which declare a FID layer creation option)

Mutually exclusive with *--schema* and with *--like*.

Standard Options

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Create a POINT layer named names with a string field named name in a new vector dataset

```
gdal vector create --geometry-type point  --crs EPSG:4326 --field name:string --output-
↪layer names points.gpkg
```

Example 2: Add a POINT layer named names2 with a string field named name to an existing vector dataset

```
gdal vector create --update --geometry-type point --crs EPSG:4326 --field name:string --
↪output-layer names2 points.gpkg
```

Example 3: Create a new vector dataset with a layer named countries_new based on the layer countries of an existing dataset

```
gdal vector create --like ../data/poly.gpkg --input-layer poly --output-layer areas_new␣
↪areas.gpkg
```

5.1.4.13 gdal vector dissolve

Added in version 3.13.

Synopsis

```
Usage: gdal vector dissolve [OPTIONS] <INPUT> <OUTPUT>

Dissolves multipart features

Positional arguments:
  -i, --input <INPUT>                                    Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>               Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>           Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                               Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                      Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                               Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                         Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                          Output layer name [not available␣
↪in pipelines]
  --skip-errors                                          Skip errors when writing features␣
↪[not available in pipelines]
  --active-layer <ACTIVE-LAYER>                          Set active layer (if not␣
↪specified, all)
  --active-geometry <ACTIVE-GEOMETRY>                    Geometry field name to which to␣
↪restrict the processing (if not specified, all)

Advanced Options:
```

(continues on next page)

(continued from previous page)

```
  --if, --input-format <INPUT-FORMAT>                    Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                      Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>        Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                               Upsert features (implies 'append
↪') [not available in pipelines]
                                                         Mutually exclusive with --append
```

Description

gdal vector dissolve performs a union operation on the elements of each feature's geometry. This has the following effects:

- Duplicate vertices are eliminated.
- Nodes are added where input linework intersects.
- Polygons that overlap are "dissolved" into a single feature.

For linear geometries, the union step is followed by a line-merging step, where lines are merged at points that form an endpoint of exactly two lines.

To dissolve the geometries of multiple features together, first combine them into single features with *gdal vector combine*.

`dissolve` can be used as a step of *gdal vector pipeline*.

Standard Options

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--active-layer <ACTIVE-LAYER>

Set the active layer. When it is specified, only the layer specified by its name will be subject to the processing. Other layers will be not modified. If this option is not specified, all layers will be subject to the processing.

--active-geometry <ACTIVE-GEOMETRY>

Set the active geometry field from its name. When it is specified, only the specified geometry field will be subject to the processing. Other geometry fields will be not modified. If this option is not specified, all geometry fields will be subject to the processing. This option can be combined together with `--active-layer`.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Dissolve country boundaries into continent boundaries

```
gdal vector pipeline read countries.shp !
    combine --group-by CONTINENT ! \
    dissolve ! \
    write continents.shp
```

5.1.4.14 gdal vector edit

Added in version 3.11.

Synopsis

```
Usage: gdal vector edit [OPTIONS] <INPUT> <OUTPUT>

Edit metadata of a vector dataset.

Positional arguments:
  -i, --input <INPUT>                                    Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>               Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
```

(continues on next page)

(continued from previous page)

```
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>         Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                          Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                             Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                    Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                             Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                       Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                        Output layer name
  --skip-errors                                        Skip errors when writing features␣
↪[not available in pipelines]
  --active-layer <ACTIVE-LAYER>                        Set active layer (if not␣
↪specified, all)
  --geometry-type <GEOMETRY-TYPE>                      Layer geometry type
  --crs <CRS>                                          Override CRS (without␣
↪reprojection)
  --metadata <KEY>=<VALUE>                             Add/update dataset metadata item␣
↪[may be repeated]
  --unset-metadata <KEY>                               Remove dataset metadata item [may␣
↪be repeated]
  --layer-metadata <KEY>=<VALUE>                       Add/update layer metadata item␣
↪[may be repeated]
  --unset-layer-metadata <KEY>                         Remove layer metadata item [may␣
↪be repeated]
  --unset-fid                                          Unset the identifier of each␣
↪feature and the FID column name

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                  Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                    Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>      Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                             Upsert features (implies 'append
↪') [not available in pipelines]
                                                       Mutually exclusive with --append
```

Description

gdal vector edit can be used to edit metadata of a vector dataset: its CRS, the layer geometry type, dataset and layer metadata.

`edit` can also be used as a step of *gdal vector pipeline*.

Features are not modified.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

--crs <CRS>

Override CRS, without reprojecting.

The coordinate reference systems that can be passed are anything supported by the *OGRSpatialReference::SetFromUserInput()* call, which includes EPSG Projected, Geographic or Compound CRS (i.e. EPSG:4296), a well known text (WKT) CRS definition, PROJ.4 declarations, or the name of a .prj file containing a WKT CRS definition.

`null` or `none` can be specified to unset an existing CRS.

--geometry-type <GEOMETRY-TYPE>

Change the layer geometry type to be one of `GEOMETRY`, `POINT`, `LINESTRING`, `POLYGON`, `MULTIPOINT`, `MULTILINESTRING`, `MULTIPOLYGON`, `GEOMETRYCOLLECTION`, `CURVE`, `CIRCULARSTRING`, `COMPOUNDCURVE`, `SURFACE`, `CURVEPOLYGON`, `MULTICURVE`, `MULTISURFACE`, `POLYHEDRALSURFACE` or `TIN`. `Z`, `M` or `ZM` suffixes can be appended to the above values to indicate the dimensionality. Note that feature geometries themselves are not modified. Thus this option can be used to fix an inappropriate geometry type at the layer level.

--layer-metadata <KEY>=<VALUE>

Add/update metadata item, at the layer level.

--metadata <KEY>=<VALUE>

Add/update metadata item, at the dataset level.

--unset-fid

Added in version 3.12.

Can be specified to prevent the name of the source FID column and source feature IDs from being reused for the target layer. This option can for example be useful if selecting source features with a ORDER BY clause.

--unset-metadata <KEY>

Remove metadata item, at the dataset level.

--unset-layer-metadata <KEY>

Remove metadata item, at the layer level.

Standard Options

--active-layer <ACTIVE-LAYER>

Set the active layer. When it is specified, only the layer specified by its name will be subject to the processing. Other layers will be not modified. If this option is not specified, all layers will be subject to the processing.

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--input-layer <INPUT-LAYER>

Specifies the name of one or more layers to process. By default, all layers will be processed. To read and write all layers but only process a subset, use `--active-layer`.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-layer <NAME>

Specifies the name of the layer to which features will be written. If not specified, output layer names will be the same as input layer names.

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Change the CRS of a GeoPackage file (without reprojecting it) and its geometry type

```
$ gdal vector edit --crs=EPSG:4326 --geometry-type=POLYGONZM in.gpkg out.gpkg --overwrite
```

Example 2: Apply a CRS to a Shapefile

If a Shapefile is missing its associated `.prj` file, it can be created as follows:

```
$ gdal vector edit --crs=EPSG:3857 in.shp out.shp
```

5.1.4.15 gdal vector filter

Added in version 3.11.

Synopsis

```
Usage: gdal vector filter [OPTIONS] <INPUT> <OUTPUT>

Filter a vector dataset.

Positional arguments:
```

(continues on next page)

(continued from previous page)

```
  -i, --input <INPUT>                                    Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>               Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>           Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                               Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                      Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                               Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                         Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                          Output layer name [not available␣
↪in pipelines]
  --skip-errors                                          Skip errors when writing features␣
↪[not available in pipelines]
  --active-layer <ACTIVE-LAYER>                          Set active layer (if not␣
↪specified, all)
  --bbox <BBOX>                                          Bounding box as xmin,ymin,xmax,
↪ymax
  --where <WHERE>|@<filename>                            Attribute query in a restricted␣
↪form of the queries used in the SQL WHERE statement
  --update-extent                                        Update layer extent to take into␣
↪account the filter

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                    Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                      Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>        Output open options [may be␣
↪repeated] [not available in pipelines]
```

(continues on next page)

(continued from previous page)

```
  --upsert                                                Upsert features (implies 'append
↪') [not available in pipelines]
                                                          Mutually exclusive with --append
```

Description

gdal vector filter can be used to filter a vector dataset from their spatial extent or a SQL WHERE clause.

`filter` can also be used as a step of *gdal vector pipeline*.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

--bbox `<xmin>,<ymin>,<xmax>,<ymax>`

Bounds to which to filter the dataset. They are assumed to be in the CRS of the input dataset. The X and Y axis are the "GIS friendly ones", that is X is longitude or easting, and Y is latitude or northing. Note that filtering does not clip geometries to the bounding box.

--update-extent

Update layer extent to take into account the filter(s). Otherwise the layer extent will generally be the one of the source layer before applying the filter(s). Note that using this option requires doing a full scan of the filtered layers.

--where `<WHERE>|@<filename>`

Attribute query (like SQL WHERE).

Standard Options

--active-layer `<ACTIVE-LAYER>`

Set the active layer. When it is specified, only the layer specified by its name will be subject to the processing. Other layers will be not modified. If this option is not specified, all layers will be subject to the processing.

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option `<NAME>=<VALUE>`

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--input-layer <INPUT-LAYER>

Specifies the name of one or more layers to process. By default, all layers will be processed. To read and write all layers but only process a subset, use *--active-layer*.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-layer <NAME>

Specifies the name of the layer to which features will be written. If not specified, output layer names will be the same as input layer names.

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Select features from a GeoPackage file that intersect the bounding box from longitude 2, latitude 49, to longitude 3, latitude 50 in WGS 84

```
$ gdal vector filter --bbox=2,49,3,50 in.gpkg out.gpkg --overwrite
```

Example 2: Filter Shapefile features with an attribute query

Bash

PowerShell

```
gdal vector pipeline \
    ! read in.shp \
    ! filter --where "CODE IS NULL AND NAME NOT LIKE 'TEMP%'" \
    ! write out.gpkg
```

```
gdal vector pipeline `
    ! read in.shp `
    ! filter --where "CODE IS NULL AND NAME NOT LIKE 'TEMP%'" `
    ! write out.gpkg
```

5.1.4.16 gdal vector info

Added in version 3.11.

Synopsis

```
Usage: gdal vector info [OPTIONS] <INPUT>...

Return information on a vector dataset.

Positional arguments:
  -i, --dataset, --input <INPUT>                      Input vector datasets [may be␣
↪repeated] [required]

Common Options:
  -h, --help                                          Display help message and exit
  --json-usage                                        Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                              Configuration option [may be␣
↪repeated]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format. OUTPUT-
↪FORMAT=json|text
  -l, --layer, --input-layer <INPUT-LAYER>            Input layer name [may be repeated]
                                                      Mutually exclusive with --sql, --
↪fid
  --features                                          List all features (beware of RAM␣
↪consumption on large layers)
                                                      Mutually exclusive with --summary
  --summary                                           List the layer names and the␣
↪geometry type
                                                      Mutually exclusive with --features
  --limit <FEATURE-COUNT>                             Limit the number of features per␣
↪layer (implies --features)
  --sql <statement>|@<filename>                       Execute the indicated SQL␣
↪statement and return the result
                                                      Mutually exclusive with --input-
↪layer, --fid
  --where <WHERE>|@<filename>                         Attribute query in a restricted␣
↪form of the queries used in the SQL WHERE statement
  --fid <FID>                                         Feature identifier
                                                      Mutually exclusive with --input-
↪layer, --sql
  --dialect <DIALECT>                                 SQL dialect

Advanced Options:
  --oo, --open-option <KEY>=<VALUE>                   Open options [may be repeated]
  --if, --input-format <INPUT-FORMAT>                 Input formats [may be repeated]

Esoteric Options:
  --crs-format <CRS-FORMAT>                           Which format to use to report CRS.
↪ CRS-FORMAT=AUTO|WKT2|PROJJSON (default: AUTO)
```

Description

gdal vector info lists various information about a GDAL supported vector dataset, and returns them on the standard output stream when used from the command line, or in the `output` parameter when used from the API.

Starting with GDAL 3.12, **gdal vector info** can be used as the last step of a pipeline.

The following options are available:

Program-Specific Options

--dialect <dialect>

SQL dialect. In some cases can be used to use (unoptimized) *OGR SQL dialect* instead of the native SQL of an RDBMS by passing the `OGRSQL` dialect value. The *SQL SQLite dialect* can be selected with the `SQLITE` and `INDIRECT_SQLITE` dialect values, and this can be used with any datasource.

--features

List all features by default, unless limited with *--limit*. Beware of RAM consumption on large layers when using JSON output. This option is mutually exclusive with the *--summary* option.

-f, --of, --format, --output-format `json|text`

Which output format to use. Default is JSON, and starting with GDAL 3.12, text when invoked from command line.

-l, --layer, --input-layer <INPUT-LAYER>

Name of one or more layers to inspect. If no layer names are passed and *--sql* is not specified, then all layers will be selected.

--limit <FEATURE-COUNT>

Added in version 3.12.

Limit the number of features reported per layer. When set, this implies *--features*.

--sql <statement>|@<filename>

Execute the indicated SQL statement and return the result. The `@<filename>` syntax can be used to indicate that the content is in the pointed filename (e.g `@my_select.txt` where my_select.txt is a file in the current directory). Data can also be edited with SQL INSERT, UPDATE, DELETE, DROP TABLE, ALTER TABLE etc if the dataset is opened in update mode. Editing capabilities depend on the selected dialect with *--dialect*.

This option is mutually exclusive with the *--where* option.

--summary

Print a summary with the list of layers and the geometry type of each layer. This option is mutually exclusive with the *--features* option.

--where <WHERE>|@<filename>

An attribute query in a restricted form of the queries used in the SQL *WHERE* statement. Only features matching the attribute query will be reported. The `@<filename>` syntax can be used to indicate that the content is in the pointed filename.

Example of `--where` and quoting:

```
--where "\"Corner Point Identifier\" LIKE '%__00_00'"
```

This option is mutually exclusive with the *--sql* option.

--fid <FID>

Added in version 3.13.

Feature identifier. Only the feature with the specified FID will be reported.

--crs-format AUTO|WKT2|PROJJSON

Added in version 3.13.

Which format to use to report the CRS. In AUTO default mode, if the CRS can be captured with an authority name and code (known of PROJ), only a summary of the CRS, including its name, ID, type and area of use will be reported. Otherwise a full WKT2:2019 definition will be reported.

> **Note**
>
> *--crs-format* can only be set when *--output-format* is set to `text`. The JSON text format includes both WKT2 and PROJJSON.

Standard Options

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Getting information on the file poly.gpkg (with text output), listing all features

```
$ gdal vector info --features poly.gpkg
INFO: Open of `poly.gpkg'
      using driver `GPKG' successful.

Layer name: poly
Metadata:
  DBF_DATE_LAST_UPDATE=2018-08-02
Geometry: Polygon
Feature Count: 10
Extent: (478315.531250, 4762880.500000) - (481645.312500, 4765610.500000)
Layer Coordinate Reference System:
  - name: OSGB36 / British National Grid
  - ID: EPSG:27700
  - type: Projected
  - projection type: British National Grid, Transverse Mercator
  - units: metre
  - area of use: United Kingdom (UK)..., west -9.01, south 49.75, east 2.01, north 61.01
```

(continues on next page)

(continued from previous page)

```
Data axis to CRS axis mapping: 1,2
FID Column = fid
Geometry Column = geom
AREA: Real (0.0)
EAS_ID: Integer64 (0.0)
PRFEDEA: String (16.0)
OGRFeature(poly):1
  AREA (Real) = 215229.266
  EAS_ID (Integer64) = 168
  PRFEDEA (String) = 35043411
  POLYGON ((479819.84375 4765180.5,479690.1875 4765259.5,479647.0 4765369.5,479730.375␣
↪4765400.5,480039.03125 4765539.5,480035.34375 4765558.5,480159.78125 4765610.5,480202.
↪28125 4765482.0,480365.0 4765015.5,480389.6875 4764950.0,480133.96875 4764856.5,480080.
↪28125 4764979.5,480082.96875 4765049.5,480088.8125 4765139.5,480059.90625 4765239.5,
↪480019.71875 4765319.5,479980.21875 4765409.5,479909.875 4765370.0,479859.875 4765270.
↪0,479819.84375 4765180.5))

OGRFeature(poly):2
  AREA (Real) = 247328.172
  EAS_ID (Integer64) = 179
  PRFEDEA (String) = 35043423
  POLYGON ((480035.34375 4765558.5,480039.03125 4765539.5,479730.375 4765400.5,479647.0␣
↪4765369.5,479690.1875 4765259.5,479819.84375 4765180.5,479779.84375 4765109.5,479681.
↪78125 4764940.0,479468.0 4764942.5,479411.4375 4764940.5,479353.0 4764939.5,479208.
↪65625 4764882.5,479196.8125 4764879.0,479123.28125 4765015.0,479046.53125 4765117.0,
↪479029.71875 4765110.5,479014.9375 4765147.5,479149.9375 4765200.5,479639.625 4765399.
↪5,480035.34375 4765558.5))

OGRFeature(poly):3
  AREA (Real) = 261752.781
  EAS_ID (Integer64) = 171
  PRFEDEA (String) = 35043414
  POLYGON ((479819.84375 4765180.5,479859.875 4765270.0,479909.875 4765370.0,479980.
↪21875 4765409.5,480019.71875 4765319.5,480059.90625 4765239.5,480088.8125 4765139.5,
↪480082.96875 4765049.5,480000.28125 4765043.0,479934.96875 4765020.0,479895.125␣
↪4765000.0,479734.375 4764865.0,479680.28125 4764852.0,479644.78125 4764827.5,479637.
↪875 4764803.0,479617.21875 4764760.0,479587.28125 4764718.0,479548.03125 4764693.5,
↪479504.90625 4764609.5,479239.8125 4764505.0,479117.8125 4764847.0,479196.8125 4764879.
↪0,479208.65625 4764882.5,479353.0 4764939.5,479411.4375 4764940.5,479468.0 4764942.5,
↪479681.78125 4764940.0,479779.84375 4765109.5,479819.84375 4765180.5))

OGRFeature(poly):4
  AREA (Real) = 547597.188
  EAS_ID (Integer64) = 173
  PRFEDEA (String) = 35043416
  POLYGON ((479014.9375 4765147.5,479029.71875 4765110.5,479117.8125 4764847.0,479239.
↪8125 4764505.0,479305.875 4764361.0,479256.03125 4764314.5,479220.90625 4764212.5,
↪479114.5 4764174.0,479018.28125 4764418.5,478896.9375 4764371.0,478748.8125 4764308.5,
↪478503.03125 4764218.0,478461.75 4764337.5,478443.9375 4764400.5,478447.8125 4764454.0,
↪478448.6875 4764531.5,478502.1875 4764541.5,478683.0 4764730.5,478621.03125 4764788.5,
↪478597.34375 4764766.5,478532.5 4764695.5,478460.125 4764615.0,478408.0625 4764654.0,
↪478315.53125 4764876.0,478889.25 4765100.0,479014.9375 4765147.5))
```

(continues on next page)

(continued from previous page)

```
OGRFeature(poly):5
  AREA (Real) = 15775.758
  EAS_ID (Integer64) = 172
  PRFEDEA (String) = 35043415
  POLYGON ((479029.71875 4765110.5,479046.53125 4765117.0,479123.28125 4765015.0,479196.
↪8125 4764879.0,479117.8125 4764847.0,479029.71875 4765110.5))

OGRFeature(poly):6
  AREA (Real) = 101429.977
  EAS_ID (Integer64) = 169
  PRFEDEA (String) = 35043412
  POLYGON ((480082.96875 4765049.5,480080.28125 4764979.5,480133.96875 4764856.5,479968.
↪46875 4764788.0,479750.6875 4764702.0,479735.90625 4764752.0,479640.09375 4764721.0,
↪479658.59375 4764670.0,479504.90625 4764609.5,479548.03125 4764693.5,479587.28125␣
↪4764718.0,479617.21875 4764760.0,479637.875 4764803.0,479644.78125 4764827.5,479680.
↪28125 4764852.0,479734.375 4764865.0,479895.125 4765000.0,479934.96875 4765020.0,
↪480000.28125 4765043.0,480082.96875 4765049.5))

OGRFeature(poly):7
  AREA (Real) = 268597.625
  EAS_ID (Integer64) = 166
  PRFEDEA (String) = 35043409
  POLYGON ((480389.6875 4764950.0,480537.15625 4765014.0,480567.96875 4764918.0,480605␣
↪4764835,480701.0625 4764738.0,480710.25 4764690.5,480588.59375 4764740.5,480540.71875␣
↪4764741.0,480515.125 4764695.0,480731.65625 4764561.5,480692.1875 4764453.5,480677.
↪84375 4764439.0,480655.34375 4764397.5,480584.375 4764353.0,480500.40625 4764326.5,
↪480358.53125 4764277.0,480192.3125 4764183.0,480157.125 4764266.5,480234.3125 4764304.
↪0,480289.125 4764348.5,480316 4764395,480343.5625 4764477.0,480343.71875 4764532.5,
↪480258.03125 4764767.0,480177.15625 4764742.0,480093.75 4764703.0,480011.0 4764674.5,
↪479985.0625 4764732.0,479968.46875 4764788.0,480133.96875 4764856.5,480389.6875␣
↪4764950.0))

OGRFeature(poly):8
  AREA (Real) = 1634833.375
  EAS_ID (Integer64) = 158
  PRFEDEA (String) = 35043369
  POLYGON ((480701.0625 4764738.0,480761.46875 4764778.0,480824.96875 4764820.0,480922.
↪03125 4764850.5,480930.71875 4764852.0,480984.25 4764875.0,481088.1875 4764936.0,
↪481136.84375 4764994.5,481281.3125 4764876.5,481291.09375 4764810.0,481465.90625␣
↪4764872.5,481457.375 4764937.0,481509.65625 4764967.0,481538.90625 4764982.5,481575.0␣
↪4764999.5,481602.125 4764915.5,481629.84375 4764829.5,481645.3125 4764797.5,481635.
↪96875 4764795.5,481235.3125 4764650.0,481209.8125 4764633.5,481199.21875 4764623.5,
↪481185.5 4764607.0,481159.9375 4764580.0,481140.46875 4764510.5,481141.625 4764480.5,
↪481199.84375 4764180.0,481143.4375 4764010.5,481130.3125 4763979.5,481039.9375 4763889.
↪5,480882.6875 4763670.0,480826.0625 4763650.5,480745.1875 4763628.5,480654.4375␣
↪4763627.5,480599.8125 4763660.0,480281.9375 4763576.5,480221.5 4763533.5,480199.6875␣
↪4763509.0,480195.09375 4763430.0,480273.6875 4763305.5,480309.6875 4763063.5,480201.
↪84375 4762962.5,479855.3125 4762880.5,479848.53125 4762897.0,479728.875 4763217.5,
↪479492.6875 4763850.0,479550.0625 4763919.5,480120.21875 4764188.5,480192.3125 4764183.
↪0,480358.53125 4764277.0,480500.40625 4764326.5,480584.375 4764353.0,480655.34375␣
↪4764397.5,480677.84375 4764439.0,480692.1875 4764453.5,480731.65625 4764561.5,480515.
```

(continues on next page)

(continued from previous page)

```
↪125 4764695.0,480540.71875 4764741.0,480588.59375 4764740.5,480710.25 4764690.5,480701.
↪0625 4764738.0))

OGRFeature(poly):9
  AREA (Real) = -596610.313
  EAS_ID (Integer64) = 165
  PRFEDEA (String) = 35043408
  POLYGON ((479750.6875 4764702.0,479968.46875 4764788.0,479985.0625 4764732.0,480011.0␣
↪4764674.5,480093.75 4764703.0,480177.15625 4764742.0,480258.03125 4764767.0,480343.
↪71875 4764532.5,480343.5625 4764477.0,480316 4764395,480289.125 4764348.5,480234.3125␣
↪4764304.0,480157.125 4764266.5,480192.3125 4764183.0,480120.21875 4764188.5,479550.
↪0625 4763919.5,479492.6875 4763850.0,479487.75 4763864.5,479442.75 4763990.0,479436␣
↪4764023,479398.9375 4764100.0,479349.625 4764230.0,479305.875 4764361.0,479239.8125␣
↪4764505.0,479504.90625 4764609.5,479658.59375 4764670.0,479750.6875 4764702.0))

OGRFeature(poly):10
  AREA (Real) = 5268.813
  EAS_ID (Integer64) = 170
  PRFEDEA (String) = 35043413
  POLYGON ((479750.6875 4764702.0,479658.59375 4764670.0,479640.09375 4764721.0,479735.
↪90625 4764752.0,479750.6875 4764702.0))
```

Example 2: Getting information on the file poly.gpkg (with JSON output)

```
$ gdal vector info --format=JSON poly.gpkg
{
  "description":"poly.gpkg",
  "driverShortName":"GPKG",
  "driverLongName":"GeoPackage",
  "layers":[
    {
      "name":"poly",
      "metadata":{
        "":{
          "DBF_DATE_LAST_UPDATE":"2018-08-02"
        }
      },
      "geometryFields":[
        {
          "name":"geom",
          "type":"Polygon",
          "nullable":true,
          "extent":[
            478315.53125,
            4762880.5,
            481645.3125,
            4765610.5
          ],
          "coordinateSystem":{
            "wkt":"PROJCRS[\"OSGB36 / British National Grid\",\n    BASEGEOGCRS[\"OSGB36\
↪",\n        DATUM[\"Ordnance Survey of Great Britain 1936\",\n            ELLIPSOID[\
```

(continues on next page)

(continued from previous page)

```
↪"Airy 1830\",6377563.396,299.3249646,\n                LENGTHUNIT[\"metre\",1]]],\n  ␣
↪    PRIMEM[\"Greenwich\",0,\n            ANGLEUNIT[\"degree\",0.0174532925199433]],\n ␣
↪      ID[\"EPSG\",4277]],\n    CONVERSION[\"British National Grid\",\n        METHOD[\
↪"Transverse Mercator\",\n            ID[\"EPSG\",9807]],\n        PARAMETER[\"Latitude␣
↪of natural origin\",49,\n            ANGLEUNIT[\"degree\",0.0174532925199433],\n      ␣
↪     ID[\"EPSG\",8801]],\n        PARAMETER[\"Longitude of natural origin\",-2,\n     ␣
↪      ANGLEUNIT[\"degree\",0.0174532925199433],\n            ID[\"EPSG\",8802]],\n    ␣
↪   PARAMETER[\"Scale factor at natural origin\",0.9996012717,\n            SCALEUNIT[\
↪"unity\",1],\n            ID[\"EPSG\",8805]],\n        PARAMETER[\"False easting\",
↪400000,\n            LENGTHUNIT[\"metre\",1],\n            ID[\"EPSG\",8806]],\n      ␣
↪ PARAMETER[\"False northing\",-100000,\n            LENGTHUNIT[\"metre\",1],\n        ␣
↪   ID[\"EPSG\",8807]]],\n    CS[Cartesian,2],\n        AXIS[\"(E)\",east,\n           ␣
↪ORDER[1],\n            LENGTHUNIT[\"metre\",1]],\n        AXIS[\"(N)\",north,\n       ␣
↪    ORDER[2],\n            LENGTHUNIT[\"metre\",1]],\n    USAGE[\n        SCOPE[\
↪"Engineering survey, topographic mapping.\"],\n        AREA[\"United Kingdom (UK) -␣
↪offshore to boundary of UKCS within 49°45'N to 61°N and 9°W to 2°E; onshore Great␣
↪Britain (England, Wales and Scotland). Isle of Man onshore.\"],\n        BBOX[49.75,-9.
↪01,61.01,2.01]],\n    ID[\"EPSG\",27700]]",
            "projjson":{
              "$schema":"https://proj.org/schemas/v0.7/projjson.schema.json",
              "type":"ProjectedCRS",
              "name":"OSGB36 / British National Grid",
              "base_crs":{
                "name":"OSGB36",
                "datum":{
                  "type":"GeodeticReferenceFrame",
                  "name":"Ordnance Survey of Great Britain 1936",
                  "ellipsoid":{
                    "name":"Airy 1830",
                    "semi_major_axis":6377563.396,
                    "inverse_flattening":299.3249646
                  }
                },
                "coordinate_system":{
                  "subtype":"ellipsoidal",
                  "axis":[
                    {
                      "name":"Geodetic latitude",
                      "abbreviation":"Lat",
                      "direction":"north",
                      "unit":"degree"
                    },
                    {
                      "name":"Geodetic longitude",
                      "abbreviation":"Lon",
                      "direction":"east",
                      "unit":"degree"
                    }
                  ]
                },
                "id":{
                  "authority":"EPSG",
```

(continues on next page)

(continued from previous page)


```
                "code":4277
              }
            },
            "conversion":{
              "name":"British National Grid",
              "method":{
                "name":"Transverse Mercator",
                "id":{
                  "authority":"EPSG",
                  "code":9807
                }
              },
              "parameters":[
                {
                  "name":"Latitude of natural origin",
                  "value":49,
                  "unit":"degree",
                  "id":{
                    "authority":"EPSG",
                    "code":8801
                  }
                },
                {
                  "name":"Longitude of natural origin",
                  "value":-2,
                  "unit":"degree",
                  "id":{
                    "authority":"EPSG",
                    "code":8802
                  }
                },
                {
                  "name":"Scale factor at natural origin",
                  "value":0.9996012717,
                  "unit":"unity",
                  "id":{
                    "authority":"EPSG",
                    "code":8805
                  }
                },
                {
                  "name":"False easting",
                  "value":400000,
                  "unit":"metre",
                  "id":{
                    "authority":"EPSG",
                    "code":8806
                  }
                },
                {
                  "name":"False northing",
                  "value":-100000,
```


(continues on next page)

(continued from previous page)


```
                    "unit":"metre",
                    "id":{
                      "authority":"EPSG",
                      "code":8807
                    }
                  }
                ]
              },
              "coordinate_system":{
                "subtype":"Cartesian",
                "axis":[
                  {
                    "name":"Easting",
                    "abbreviation":"E",
                    "direction":"east",
                    "unit":"metre"
                  },
                  {
                    "name":"Northing",
                    "abbreviation":"N",
                    "direction":"north",
                    "unit":"metre"
                  }
                ]
              },
              "scope":"Engineering survey, topographic mapping.",
              "area":"United Kingdom (UK) - offshore to boundary of UKCS within 49°45'N␣
↪to 61°N and 9°W to 2°E; onshore Great Britain (England, Wales and Scotland). Isle of␣
↪Man onshore.",
              "bbox":{
                "south_latitude":49.75,
                "west_longitude":-9.01,
                "north_latitude":61.01,
                "east_longitude":2.01
              },
              "id":{
                "authority":"EPSG",
                "code":27700
              }
            },
            "dataAxisToSRSAxisMapping":[
              1,
              2
            ]
          }
        }
      ],
      "featureCount":10,
      "fidColumnName":"fid",
      "fields":[
        {
          "name":"AREA",
```


(continues on next page)

(continued from previous page)

```
          "type":"Real",
          "nullable":true,
          "uniqueConstraint":false
        },
        {
          "name":"EAS_ID",
          "type":"Integer64",
          "nullable":true,
          "uniqueConstraint":false
        },
        {
          "name":"PRFEDEA",
          "type":"String",
          "width":16,
          "nullable":true,
          "uniqueConstraint":false
        }
      ]
    }
  ],
  "metadata":{
  },
  "domains":{
  },
  "relationships":{
  }
}
```

Example 3: List all layers in a dataset using jq

```
gdal vector info av_2056.gpkg --format json | jq ".layers[].name"
```

Example 4: List all layers and their geometry fields using jq

Bash

PowerShell

```
$ gdal vector info counties-albers-10m.gpkg --output-format=JSON | jq -r '
  .layers[]
  | .name as $layer
  | .geometryFields[]?
  | "\($layer): \(.name) (\(.type))"'
```

```
gdal vector info counties-albers-10m.gpkg --output-format=JSON | jq -r '
  .layers[]
  | .name as $layer
  | .geometryFields[]?
  | \"\($layer): \(.name) (\(.type))\"'
```

returns:

```
counties: geom (Geometry)
states: geom (MultiPolygon)
nation: geom (MultiPolygon)
```

5.1.4.17 gdal vector export-schema

Added in version 3.13.

Synopsis

```
Usage: gdal vector export-schema [OPTIONS] <INPUT>...

Export the OGR_SCHEMA from a vector dataset.

Positional arguments:
  -i, --dataset, --input <INPUT>            Input vector datasets [may be repeated]␣
↪[required]

Common Options:
  -h, --help                                Display help message and exit
  --json-usage                              Display usage as JSON document and exit
  --config <KEY>=<VALUE>                    Configuration option [may be repeated]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>  Input layer name [may be repeated]
  -o, --output <OUTPUT>                     Output file name. If not specified, output␣
↪is sent to stdout
  --overwrite                               Whether overwriting existing output file is␣
↪allowed

Advanced Options:
  --oo, --open-option <KEY>=<VALUE>         Open options [may be repeated]
  --if, --input-format <INPUT-FORMAT>       Input formats [may be repeated]
```

Description

gdal vector export-schema exports the OGR_SCHEMA from a GDAL-supported vector dataset, and returns it on the standard output stream when used from the command line, or in the output parameter when used from the API.

OGR_SCHEMA is a JSON object describing the structure of a vector dataset according to the schema definition at ogr_fields_override.schema.json

gdal vector export-schema can be used as the last step of a pipeline.

The following options are available:

Program-Specific Options

-l, --layer, --input-layer <INPUT-LAYER>

Name of one or more layers to inspect. If no layer names are passed, then all layers will be selected.

Standard Options

-o, --output <OUTPUT>

Added in version 3.14.

Filename to write the OGR_SCHEMA. If not specified, the output will be written to the standard output stream.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Extracting the OGR_SCHEMA from the file poly.gpkg

```
gdal vector export-schema poly.gpkg
```

Example 2: Save the OGR_SCHEMA to a file

When using Windows PowerShell 5.1, redirecting output with > or `Out-File -Encoding utf8` produces UTF-8 with a BOM (Byte Order Mark). When the schema is read by *gdal vector create*, this may result in `ERROR 1: JSON parsing error: unexpected character (at offset 0)`.

To avoid encoding issues, write the file using the *--output* option.

Bash

PowerShell

```
gdal vector export-schema natural_earth_vector.gpkg --layer "ne_50m_admin_0_countries" >␣
↪countries.json
```

```
# PowerShell (all versions)
gdal vector export-schema natural_earth_vector.gpkg --layer "ne_50m_admin_0_countries" --
↪output countries.json --overwrite

# PowerShell 7+ only
gdal vector export-schema natural_earth_vector.gpkg --layer "ne_50m_admin_0_countries" |␣
↪Out-File countries.json -Encoding utf8
```

Example 3: Validate an OGR_SCHEMA file using Python

Validate against the latest version of the schema definition at ogr_fields_override.schema.json using the Python package check-jsonschema available on PyPI.

```
$ pip install check-jsonschema
$ check-jsonschema --schemafile https://raw.githubusercontent.com/OSGeo/gdal/master/ogr/
↪data/ogr_fields_override.schema.json countries.json --verbose
```

Example 4: Create an OGR_SCHEMA at the end of a pipeline

This example renames the field `pop_est` to `estimated_population` and changes its type from `Real` to `Integer`, before exporting the resulting OGR_SCHEMA.

Bash

PowerShell

```
gdal vector pipeline \
    ! read natural_earth_vector.gpkg --layer ne_50m_admin_0_countries \
    ! sql --sql "SELECT geom,name,abbrev,pop_est AS estimated_population FROM ne_50m_
↪admin_0_countries" --output-layer "countries" \
    ! set-field-type --field-name "estimated_population" --field-type Integer \
    ! export-schema --output countries-population.json
```

```
gdal vector pipeline `
    ! read natural_earth_vector.gpkg --layer ne_50m_admin_0_countries `
    ! sql --sql "SELECT geom,name,abbrev,pop_est AS estimated_population FROM ne_50m_
↪admin_0_countries" --output-layer "countries" `
    ! set-field-type --field-name "estimated_population" --field-type Integer `
    ! export-schema --output countries-population.json
```

5.1.4.18 gdal vector explode-collections

Added in version 3.12.

Synopsis

```
Usage: gdal vector explode-collections [OPTIONS] <INPUT> <OUTPUT>

Explode geometries of type collection of a vector dataset.

Positional arguments:
  -i, --input <INPUT>                                    Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
```

(continues on next page)

(continued from previous page)

```
  -q, --quiet                                           Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>              Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>   Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                 Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>          Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                           Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                              Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                     Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                              Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                        Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                         Output layer name [not available␣
↪in pipelines]
  --skip-errors                                         Skip errors when writing features␣
↪[not available in pipelines]
  --active-layer <ACTIVE-LAYER>                         Set active layer (if not␣
↪specified, all)
  --active-geometry <ACTIVE-GEOMETRY>                   Geometry field name to which to␣
↪restrict the processing (if not specified, all)
  --geometry-type <GEOMETRY-TYPE>                       Geometry type
  --skip-on-type-mismatch                               Skip feature when change of␣
↪feature geometry type failed

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                   Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                     Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>       Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                              Upsert features (implies 'append
↪') [not available in pipelines]
                                                        Mutually exclusive with --append
```

Description

gdal vector explode-collections produces one feature for each geometry in any kind of geometry collection.

For example if a feature contains a geometry `MULTIPOINT(1 2,3 4)`, two corresponding target features will be generated: one with a geometry `POINT(1 2)` and another one with geometry `POINT(3 4)`. Note that collections are not recursively exploded, i.e. `GEOMETRYCOLLECTION(GEOMETRYCOLLECTION(POINT (1 2),POINT(3 4))` will be exploded as `GEOMETRYCOLLECTION(POINT (1 2),POINT(3 4))`.

It can also be used as a step of *gdal vector pipeline*.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

--geometry-type <GEOMETRY-TYPE>

Change the geometry type of exploded geometries to be one of `GEOMETRY`, `POINT`, `LINESTRING`, `POLYGON`, `CIRCULARSTRING`, `COMPOUNDCURVE`, `CURVEPOLYGON`, `POLYHEDRALSURFACE` or `TIN`. `Z`, `M` or `ZM` suffixes can be appended to the above values to indicate the dimensionality. The presence of geometries that cannot be coerced to the specified type will cause an error, unless *--skip-on-type-mismatch* is specified.

--skip-on-type-mismatch

Do not generate output features when their target geometry type does not match the one specified by *--geometry-type*.

Standard Options

--active-geometry <ACTIVE-GEOMETRY>

Set the active geometry field from its name. When it is specified, only the specified geometry field will be subject to the processing. Other geometry fields will be not modified. If this option is not specified, all geometry fields will be subject to the processing. This option can be combined together with *--active-layer*.

--active-layer <ACTIVE-LAYER>

Set the active layer. When it is specified, only the layer specified by its name will be subject to the processing. Other layers will be not modified. If this option is not specified, all layers will be subject to the processing.

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--input-layer <INPUT-LAYER>

Specifies the name of one or more layers to process. By default, all layers will be processed. To read and write all layers but only process a subset, use *--active-layer*.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-layer <NAME>

Specifies the name of the layer to which features will be written. If not specified, output layer names will be the same as input layer names.

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Only retain parts of geometry collections that are of type Point.

```
$ gdal vector explode-collections --geometry-type=POINT --skip-on-type-mismatch in.gpkg␣
↪points.shp --overwrite
```

5.1.4.19 gdal vector grid

Added in version 3.11.

Synopsis

```
Usage: gdal vector grid <SUBCOMMAND> [OPTIONS]
where <SUBCOMMAND> is one of:
  - average:                Create a regular grid from scattered points using moving␣
↪average interpolation.
  - average-distance:       Create a regular grid from scattered points using the␣
↪average distance between the grid node (center of the search ellipse) and all of the␣
↪data points in the search ellipse.
  - average-distance-points: Create a regular grid from scattered points using the␣
↪average distance between the data points in the search ellipse.
  - count:                  Create a regular grid from scattered points using the␣
↪number of points in the search ellipse.
  - invdist:                Create a regular grid from scattered points using weighted␣
↪inverse distance interpolation.
  - invdistnn:              Create a regular grid from scattered points using weighted␣
↪inverse distance interpolation nearest neighbour.
  - linear:                 Create a regular grid from scattered points using linear/
↪barycentric interpolation.
  - maximum:                Create a regular grid from scattered points using the␣
↪maximum value in the search ellipse.
  - minimum:                Create a regular grid from scattered points using the␣
↪minimum value in the search ellipse.
  - nearest:                Create a regular grid from scattered points using nearest␣
↪neighbor interpolation.
  - range:                  Create a regular grid from scattered points using the␣
```

(continues on next page)

(continued from previous page)

```
↪difference between the minimum and maximum values in the search ellipse.

Common Options:
  -h, --help              Display help message and exit
  --json-usage            Display usage as JSON document and exit
  --config <KEY>=<VALUE>  Configuration option [may be repeated]
```

Description

This program creates a regular grid (raster) from the scattered data read from a vector dataset. Input data will be interpolated to fill grid nodes with values, you can choose from various interpolation methods.

It is possible to set the *GDAL_NUM_THREADS* configuration option to parallelize the processing. The value to specify is the number of worker threads, or `ALL_CPUS` to use all the cores/CPUs of the computer.

Since GDAL 3.12, this algorithm can be part of a *gdal pipeline*.

Options common to all algorithms

Standard options

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep + | sort`

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

-l, --layer, --layer-name <LAYER-NAME>

Indicates the layer(s) from the datasource that will be used for input features. May be specified multiple times

--sql <SQL>|@<filename>

An SQL statement to be evaluated against the datasource to produce a virtual layer of features to be burned in. The @filename syntax can be used to indicate that the content is in the pointed filename.

--extent <xmin>,<ymin>,<xmax>,<ymax>

Set georeferenced extents. The values must be expressed in georeferenced units. If not specified, the extent of the output file will be the extent of the vector layers.

--resolution <xres>,<yres>

Set target resolution. The values must be expressed in georeferenced units. Both must be positive values. Note that *--resolution* cannot be used with *--size*.

--size <xsize>,<ysize>

Set output file size in pixels and lines. Note that *--size* cannot be used with *--resolution*.

--ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>

Force the output bands to be of the indicated data type. Defaults to `Float64`.

--crs <CRS>

Override the projection for the output file. If not specified, the projection of the input vector file will be used if available. When using this option, no reprojection of features from the CRS of the input vector to the specified CRS of the output raster, so use only this option to correct an invalid source CRS. The <CRS> may be any of the usual GDAL/OGR forms, complete WKT, PROJ.4, EPSG:n or a file containing the WKT.

--bbox <xmin>,<ymin>,<xmax>,<ymax>

Select only points contained within the specified bounding box.

--zfield <field_name>

Identifies an attribute field on the features to be used to get a Z value from. This value overrides the Z value read from the feature geometry record (naturally, if you have a Z value in the geometry, otherwise you have no choice and should specify a field name containing a Z value).

--zincrease <increase_value>

Addition to the attribute field on the features to be used to get a Z value from. The addition should be the same unit as the Z value. The result value will be Z value + Z increase value. The default value is 0.

--zmultiply <multiply_value>

This is multiplication ratio for the Z field. This can be used for a shift from e.g. feet to meters or from elevation to depth. The result value will be (Z value + Z increase value) * Z multiply value. The default value is 1.

Advanced options

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

"invdist" algorithm

Interpolation using an inverse distance to a power.

When *--min-points-per-quadrant* or *--max-points-per-quadrant* is specified, the actual algorithm used is "invdistnn".

Options

--power <val>

Weighting power (default 2.0).

--smoothing <val>

Smoothing parameter (default 0.0).

--radius <val>

Set first and second radius (mutually exclusive with *--radius1* and *--radius2*. By default, uses the whole point array.

--radius1 <val>

The first radius (X axis if rotation angle is 0) of the search ellipse. By default, uses the whole point array.

--radius2 <val>

The second radius (Y axis if rotation angle is 0) of the search ellipse. By default, uses the whole point array.

--angle <val>

Angle of search ellipse rotation in degrees (counter clockwise, default is 0).

--max-points <val>

Maximum number of data points to use. Do not search for more points than this number. This may only used if the search ellipse is set (both radii are non-zero). By default, no limitation.

--min-points <val>

Minimum number of data points to use. If less points in the search ellipse than the specified value are found, the grid node considered empty and will be filled with the nodata value. This may only used if the search ellipse is set (both radii are non-zero). By default, no limitation.

--max-points-per-quadrant <val>

Maximum number of data points to use per quadrant. By default, no limitation. When specified, the algorithm will only take into account up to max_points_per_quadrant points for each of the right-top, left-top, right-bottom and right-top quadrant relative to the point being interpolated.

--min-points-per-quadrant <val>

Minimum number of data points to use per quadrant. By default, no limitation. When specified, the algorithm will collect at least min_points_per_quadrant points for each of the right-top, left-top, right-bottom and right-top quadrant relative to the point being interpolated.

--nodata <val>

Nodata value to fill empty points (default is 0).

"invdistnn" algorithm

Interpolation using an inverse distance to a power with nearest neighbor searching, ideal when *--max-points* is used.

When *--min-points-per-quadrant* or *--max-points-per-quadrant* is specified, the search will start with the closest point to the point being interpolated from the first quadrant, then the closest point to the point being interpolated from the second quadrant, etc. up to the 4th quadrant, and will continue with the next closest point in the first quadrant, etc. until *--max-points* and/or *--max-points-per-quadrant* thresholds are reached.

Required option

--radius <val>

Set search radius (mutually exclusive with *--radius1* and *--radius2*. Required.

Options

--power <val>

Weighting power (default 2.0).

--smoothing <val>

Smoothing parameter (default 0.0).

--max-points <val>

Maximum number of data points to use. Do not search for more points than this number. This may only used if the search ellipse is set (both radii are non-zero). By default, no limitation.

--min-points <val>

Minimum number of data points to use. If less points in the search ellipse than the specified value are found, the grid node considered empty and will be filled with the nodata value. This may only used if the search ellipse is set (both radii are non-zero). By default, no limitation.

--max-points-per-quadrant <val>

Maximum number of data points to use per quadrant. By default, no limitation. When specified, the algorithm will only take into account up to max_points_per_quadrant points for each of the right-top, left-top, right-bottom and right-top quadrant relative to the point being interpolated.

--min-points-per-quadrant <val>

Minimum number of data points to use per quadrant. By default, no limitation. When specified, the algorithm will collect at least min_points_per_quadrant points for each of the right-top, left-top, right-bottom and right-top quadrant relative to the point being interpolated.

--nodata <val>

Nodata value to fill empty points (default is 0).

"average" algorithm

Interpolation using a moving average algorithm.

When *--min-points-per-quadrant* or *--max-points-per-quadrant* is specified, the search will start with the closest point to the point being interpolated from the first quadrant, then the closest point to the point being interpolated from the second quadrant, etc. up to the 4th quadrant, and will continue with the next closest point in the first quadrant, etc. until *--max-points* and/or *--max-points-per-quadrant* thresholds are reached.

Options

--radius <val>

Set first and second radius (mutually exclusive with *--radius1* and *--radius2*. By default, uses the whole point array.

--radius1 <val>

The first radius (X axis if rotation angle is 0) of the search ellipse. By default, uses the whole point array.

--radius2 <val>

The second radius (Y axis if rotation angle is 0) of the search ellipse. By default, uses the whole point array.

--angle <val>

Angle of search ellipse rotation in degrees (counter clockwise, default is 0).

--max-points <val>

Maximum number of data points to use. Do not search for more points than this number. This may only used if the search ellipse is set (both radii are non-zero). By default, no limitation.

--min-points <val>

Minimum number of data points to use. If less points in the search ellipse than the specified value are found, the grid node considered empty and will be filled with the nodata value. This may only used if the search ellipse is set (both radii are non-zero). By default, no limitation.

--max-points-per-quadrant <val>

Maximum number of data points to use per quadrant. By default, no limitation. When specified, the algorithm will only take into account up to max_points_per_quadrant points for each of the right-top, left-top, right-bottom and right-top quadrant relative to the point being interpolated.

--min-points-per-quadrant <val>

Minimum number of data points to use per quadrant. By default, no limitation. When specified, the algorithm will collect at least min_points_per_quadrant points for each of the right-top, left-top, right-bottom and right-top quadrant relative to the point being interpolated.

--nodata <val>

Nodata value to fill empty points (default is 0).

"nearest" algorithm

Interpolation using nearest neighbor algorithm.

Options

--radius <val>

Set first and second radius (mutually exclusive with *--radius1* and *--radius2*. By default, uses the whole point array.

--radius1 <val>

The first radius (X axis if rotation angle is 0) of the search ellipse. By default, uses the whole point array.

--radius2 <val>

The second radius (Y axis if rotation angle is 0) of the search ellipse. By default, uses the whole point array.

--angle <val>

Angle of search ellipse rotation in degrees (counter clockwise, default is 0).

--nodata <val>

Nodata value to fill empty points (default is 0).

"linear" algorithm

Linear interpolation by computing a Delaunay triangulation of the point cloud, finding in which triangle of the triangulation the point is, and by doing linear interpolation from its barycentric coordinates within the triangle. If the point is not in any triangle, depending on the radius, the algorithm will use the value of the nearest point or the nodata value.

Options

--radius <val>

In case the point to be interpolated does not fit into a triangle of the Delaunay triangulation, use that maximum distance to search a nearest neighbour, or use nodata otherwise. If unset, the search distance is infinite. If set to 0, nodata value will be always used.

--nodata <val>

Nodata value to fill empty points (default is 0).

Data metrics algorithms

Besides the interpolation functionality **gdal vector grid** can be used to compute some data metrics using the specified window and output grid geometry. These metrics are:

- `minimum`: Minimum value found in grid node search ellipse.
- `maximum`: Maximum value found in grid node search ellipse.
- `range`: A difference between the minimum and maximum values found in grid node search ellipse.
- `count`: A number of data points found in grid node search ellipse.
- `average-distance`: An average distance between the grid node (center of the search ellipse) and all of the data points found in grid node search ellipse.
- `average-distance-points`: An average distance between the data points found in grid node search ellipse. The distance between each pair of points within ellipse is calculated and average of all distances is set as a grid node value.

All the metrics have the same set of options:

--radius <val>

Set first and second radius (mutually exclusive with *--radius1* and *--radius2*. By default, uses the whole point array.

--radius1 <val>

The first radius (X axis if rotation angle is 0) of the search ellipse. By default, uses the whole point array.

--radius2 <val>

The second radius (Y axis if rotation angle is 0) of the search ellipse. By default, uses the whole point array.

--angle <val>

Angle of search ellipse rotation in degrees (counter clockwise, default is 0).

--min-points <val>

Minimum number of data points to use. If less points in the search ellipse than the specified value are found, the grid node considered empty and will be filled with the nodata value. This may only used if the search ellipse is set (both radii are non-zero). By default, no limitation.

--max-points-per-quadrant <val>

Maximum number of data points to use per quadrant. By default, no limitation. When specified, the algorithm will only take into account up to max_points_per_quadrant points for each of the right-top, left-top, right-bottom and right-top quadrant relative to the point being interpolated.

--min-points-per-quadrant <val>

Minimum number of data points to use per quadrant. By default, no limitation. If that number is not reached, the point being interpolated will be set with the nodata value. When specified, the algorithm will collect at least min_points_per_quadrant points for each of the right-top, left-top, right-bottom and right-top quadrant relative to the point being interpolated.

--nodata <val>

Nodata value to fill empty points (default is 0).

When *--min-points-per-quadrant* or *--max-points-per-quadrant* is specified, the search will start with the closest point to the point being interpolated from the first quadrant, then the closest point to the point being interpolated from the second quadrant, etc. up to the 4th quadrant, and will continue with the next closest point in the first quadrant, etc. until *--max-points* and/or *--max-points-per-quadrant* thresholds are reached.

Reading comma separated values

Often you have a text file with a list of comma separated XYZ values to work with (so called CSV file). You can easily use that kind of data source in **gdal vector grid**. All you need is to create a virtual dataset header (VRT) for your CSV file and use it as an input datasource for **gdal vector grid**. You can find details on the VRT format on the *VRT -- OGR Virtual Format* description page.

Here is a small example. Let's say we have a CSV file called *dem.csv* containing

```
Easting,Northing,Elevation
86943.4,891957,139.13
87124.3,892075,135.01
86962.4,892321,182.04
87077.6,891995,135.01
...
```

For the above data we will create a *dem.vrt* header with the following content:

```
<OGRVRTDataSource>
    <OGRVRTLayer name="dem">
        <SrcDataSource>dem.csv</SrcDataSource>
        <GeometryType>wkbPoint</GeometryType>
        <GeometryField encoding="PointFromColumns" x="Easting" y="Northing" z="Elevation
↪"/>
    </OGRVRTLayer>
</OGRVRTDataSource>
```

This description specifies so called 2.5D geometry with three coordinates X, Y and Z. The Z value will be used for interpolation. Now you can use *dem.vrt* with all OGR programs (start with *ogrinfo* to test that everything works fine). The datasource will contain a single layer called *"dem"* filled with point features constructed from values in the CSV file. Using this technique you can handle CSV files with more than three columns, switch columns, etc. OK, now the final step:

```
gdal vector grid invdist dem.vrt demv.tif
```

Or, if we do not wish to use a VRT file:

```
gdal vector grid invdist -l dem -oo X_POSSIBLE_NAMES=Easting \
-oo Y_POSSIBLE_NAMES=Northing --zfield=Elevation dem.csv dem.tif
```

If your CSV file does not contain column headers then it can be handled in the VRT file in the following way:

```
<GeometryField encoding="PointFromColumns" x="field_1" y="field_2" z="field_3"/>
```

The *Comma Separated Value (.csv)* description page contains details on CSV format supported by GDAL.

Added in version 3.12.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

> **Note**
>
> However this algorithm is not natively streaming compatible. Consequently a temporary dataset will be generated, which may cause significant processing time at opening.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Create a raster from a VRT datasource using inverse distance to a power

```
gdal vector grid invdist --power=2.0 --smoothing=1.0 --extent=85000,894000,89000,890000 \
    --size=400,400 -l dem dem.vrt dem.tif
```

This example creates a raster TIFF file from the VRT datasource described in *Reading comma separated values* section using the inverse distance to a power method. Values to interpolate will be read from Z value of geometry record.

Example 2: Read values to interpolate from an attribute field

```
gdal vector grid invdist --zfield=Elevation --config GDAL_NUM_THREADS ALL_CPUS \
    --power=2.0 --smoothing=1.0 --extent=85000,894000,89000,890000 \
    --size=400,400 -l dem dem.vrt dem.tif
```

This command does the same thing as the previous one, but reads values to interpolate from the attribute field specified with `--zfield` option instead of geometry record. So in this case X and Y coordinates are being taken from geometry and Z is being taken from the *"Elevation"* field. The `GDAL_NUM_THREADS` is also set to parallelize the computation.

5.1.4.20 gdal vector index

Added in version 3.12.

Synopsis

```
Usage: gdal vector index [OPTIONS] <INPUT>... <OUTPUT>

Create a vector index of vector datasets.

Positional arguments:
  -i, --input <INPUT>                                    Input vector datasets [may be␣
↪repeated] [required]
  -o, --output <OUTPUT>                                  Output vector dataset [required]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
```

(continues on next page)

(continued from previous page)

```
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format
  --co, --creation-option <KEY>=<VALUE>                Creation option [may be repeated]
  --lco, --layer-creation-option <KEY>=<VALUE>         Layer creation option [may be␣
↪repeated]
  --overwrite                                          Whether overwriting existing␣
↪output dataset is allowed
                                                       Mutually exclusive with --update
  --update                                             Whether to open existing dataset␣
↪in update mode
                                                       Mutually exclusive with --
↪overwrite
  --overwrite-layer                                    Whether overwriting existing␣
↪output layer is allowed
  --append                                             Whether appending to existing␣
↪layer is allowed
  --nln, --output-layer <OUTPUT-LAYER>                 Output layer name
  --recursive                                          Whether input directories should␣
↪be explored recursively.
  --filename-filter <FILENAME-FILTER>                  Pattern that the filenames in␣
↪input directories should follow ('*' and '?' wildcard) [may be repeated]
  --location-name <LOCATION-NAME>                      Name of the field with the vector␣
↪path (default: location)
  --absolute-path                                      Whether the path to the input␣
↪datasets should be stored as an absolute path
  --output-crs <OUTPUT-CRS>                            Output CRS
  --metadata <KEY>=<VALUE>                             Add dataset metadata item [may be␣
↪repeated]
  --source-crs-field-name <SOURCE-CRS-FIELD-NAME>      Name of the field to store the␣
↪CRS of each dataset
  --source-crs-format <SOURCE-CRS-FORMAT>              Format in which the CRS of each␣
↪dataset must be written. SOURCE-CRS-FORMAT=auto|WKT|EPSG|PROJ (default: auto)
  --source-layer-name <SOURCE-LAYER-NAME>              Add layer of specified name from␣
↪each source file in the tile index [may be repeated]
  --source-layer-index <SOURCE-LAYER-INDEX>            Add layer of specified index (0-
↪based) from each source file in the tile index [may be repeated]
  --accept-different-crs                               Whether layers with different CRS␣
↪are accepted
  --accept-different-schemas                           Whether layers with different␣
↪schemas are accepted
  --dataset-name-only                                  Whether to write the dataset name␣
↪only, instead of suffixed with the layer index
```

Description

gdal vector index creates a vector dataset with a record for each input vector file, an attribute containing the filename (suffixed with comma and the index of the layer within the dataset) and a polygon geometry outlining the (rectangular) extent. This output is suitable for use with MapServer as a vector tileindex

Program-Specific Options

--absolute-path

The absolute path to the vector files is stored in the index file. By default the vector filenames will be put in the file exactly as they are specified on the command line.

--accept-different-crs

Whether layers with different CRS are accepted. By default, unless *--output-crs* is specified, layers that do not have the same CRS as the first index layer will be skipped. Setting *--accept-different-crs* may be useful to avoid the CRS consistency check.

--accept-different-schemas

Whether layers with different schemas are accepted. By default, layers that do not have the same schemas as the first index layer will be skipped.

--dataset-name-only

Whether to write the dataset name only, instead of suffixed with the layer index.

> **Warning**
>
> Setting this option will generate a location not compatible with MapServer.

--filename-filter `<FILENAME-FILTER>`

Pattern that the filenames contained in input directories should follow. '*' is a wildcard character that matches any number of any characters including none. '?' is a wildcard character that matches a single character. Comparisons are done in a case insensitive way. Several filters may be specified.

For example: `--filename-filter "*.shp,*.gpkg"`

--location-name `<LOCATION-NAME>`

The output field name to hold the file path/location to the indexed vectors. The default field name is `location`.

--metadata `<KEY>=<VALUE>`

Write an arbitrary layer metadata item, for formats that support layer metadata. This option may be repeated.

--output-crs `<OUTPUT-CRS>`

Envelopes of the input files will be transformed to the desired target coordinate reference system. Default creates simple rectangular polygons in the same coordinate reference system as the input vectors.

--recursive

Whether input directories should be explored recursively.

--source-crs-field-name `<SOURCE-CRS-FIELD-NAME>`

The name of the field to store the CRS of each tile. This field name can be used as the value of the TILESRS keyword in MapServer

--source-crs-format `auto|WKT|EPSG|PROJ`

The format in which the CRS of each tile must be written. Types can be `auto`, `WKT`, `EPSG`, `PROJ`. This option should be used together with *--source-crs-field-name*.

--source-layer-name `<SOURCE-LAYER-NAME>`

Add layer of specified name from each source file in the tile index [may be repeated] If none of

--source-layer-index `<SOURCE-LAYER-INDEX>`

Add layer of specified index (0-based) from each source file in the tile index [may be repeated]

Standard Options

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--nln, --output-layer <LAYER>

Provides a name for the output vector layer.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--update

Whether to open an existing output dataset in update mode.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1

Produce a GeoPackage with a record for every layer that the utility found in the `countries` folder. Each record holds information that points to the location of each input vector file and also a bounding rectangle shape showing its bounds:

```
gdal vector index countries/*.shp index.gpkg
```

Example 2

The *--output-crs* option can also be used to transform all input vector envelopes into the same output projection:

```
gdal vector index --output-crs EPSG:4326 --source-crs-field-name=src_srs countries/*.shp␣
↪index.gpkg
```

5.1.4.21 gdal vector layer-algebra

Added in version 3.12.

Synopsis

```
Usage: gdal vector layer-algebra [OPTIONS] <OPERATION> <INPUT> <METHOD> <OUTPUT>

Perform algebraic operation between 2 layers.

Positional arguments:
  --operation <OPERATION>                                Operation to perform.␣
↪OPERATION=union|intersection|sym-difference|identity|update|clip|erase [required]
  -i, --input <INPUT>                                    Input vector dataset [required]
  --method <METHOD>                                      Method vector dataset [required]
  -o, --output <OUTPUT>                                  Output vector dataset [required]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]
  --lco, --layer-creation-option <KEY>=<VALUE>           Layer creation option [may be␣
↪repeated]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed
  --update                                               Whether to open existing dataset␣
↪in update mode
  --overwrite-layer                                      Whether overwriting existing␣
↪output layer is allowed
```

(continues on next page)

(continued from previous page)

```
  --append                                          Whether appending to existing␣
↪layer is allowed
  --input-layer <INPUT-LAYER>                       Input layer name
  --method-layer <METHOD-LAYER>                     Method layer name
  --output-layer <OUTPUT-LAYER>                     Output layer name
  --geometry-type <GEOMETRY-TYPE>                   Geometry type

Advanced Options:
  --oo, --open-option <KEY>=<VALUE>                 Open options [may be repeated]
  --if, --input-format <INPUT-FORMAT>               Input formats [may be repeated]
  --input-prefix <INPUT-PREFIX>                     Prefix for fields corresponding␣
↪to input layer
  --input-field <INPUT-FIELD>                       Input field(s) to add to output␣
↪layer [may be repeated]
                                                    Mutually exclusive with --no-
↪input-field, --all-input-field
  --no-input-field                                  Do not add any input field to␣
↪output layer
                                                    Mutually exclusive with --input-
↪field, --all-input-field
  --all-input-field                                 Add all input fields to output␣
↪layer
                                                    Mutually exclusive with --input-
↪field, --no-input-field
  --method-prefix <METHOD-PREFIX>                   Prefix for fields corresponding␣
↪to method layer
  --method-field <METHOD-FIELD>                     Method field(s) to add to output␣
↪layer [may be repeated]
                                                    Mutually exclusive with --no-
↪method-field, --all-method-field
  --no-method-field                                 Do not add any method field to␣
↪output layer
                                                    Mutually exclusive with --method-
↪field, --all-method-field
  --all-method-field                                Add all method fields to output␣
↪layer
                                                    Mutually exclusive with --method-
↪field, --no-method-field
```

Description

gdal vector layer-algebra performs various vector layer algebraic operations. The command takes a vector input source and a method source and generates the output of the operation in the specified output file.

Program-Specific Options

--geometry-type <GEOMETRY-TYPE>

Change the geometry type to be one of `GEOMETRY`, `POINT`, `LINESTRING`, `POLYGON`, `MULTIPOINT`, `MULTILINESTRING`, `MULTIPOLYGON`, `GEOMETRYCOLLECTION`, `CURVE`, `CIRCULARSTRING`, `COMPOUNDCURVE`, `SURFACE`, `CURVEPOLYGON`, `MULTICURVE`, `MULTISURFACE`, `POLYHEDRALSURFACE` or `TIN`. `Z`, `M` or `ZM` suffixes can be appended to the above values to indicate the dimensionality.

--input-layer <INPUT-LAYER>

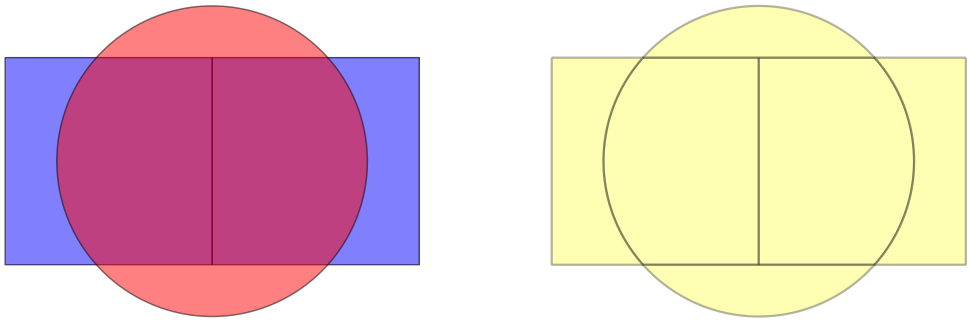
Name of the input vector layer.

--operation `union|intersection|sym-difference|identity|update|clip|erase`

Select the operation to perform among:

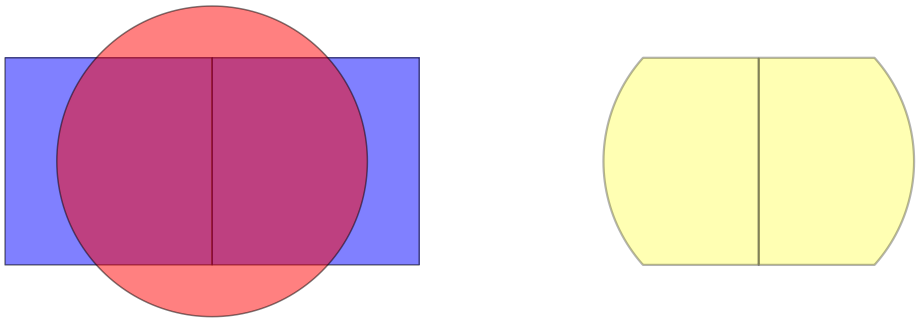
- `union`

  A union is a set of features, which represent areas that are in either of the operand layers. The operation is symmetric, and input and method layers can be interchanged.



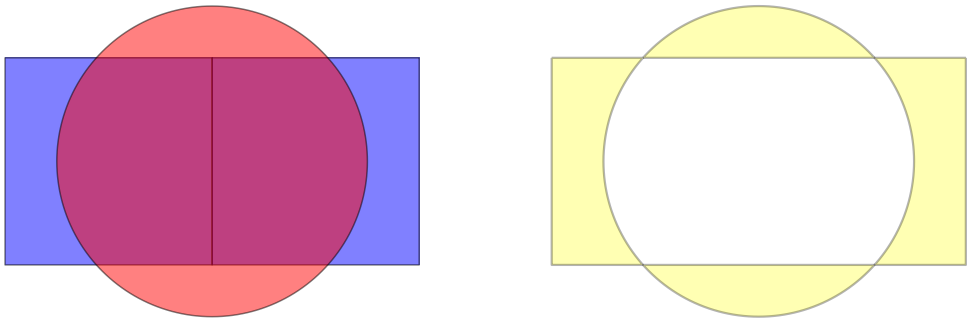
- `intersection`

  An intersection is a set of features, which represent the common areas of two layers. The operation is symmetric, and input and method layers can be interchanged.



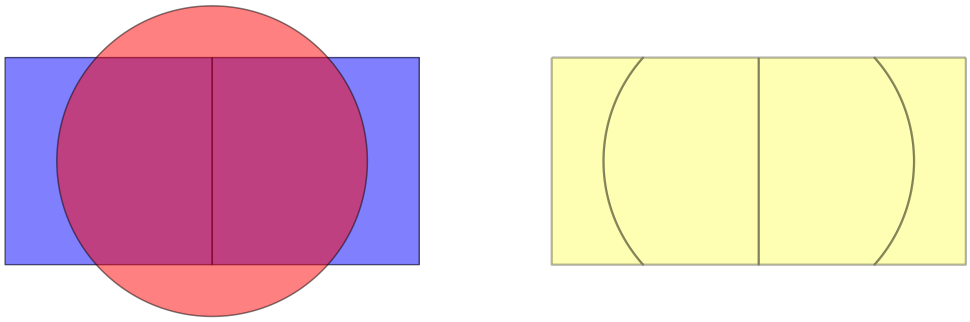
- `sym-difference`

  A symmetric difference is a set of features, which represent areas that are in operand layers but which do not intersect. The operation is symmetric, and input and method layers can be interchanged.



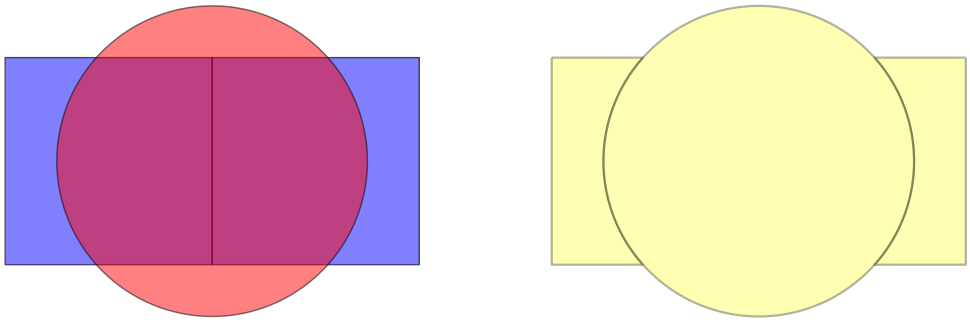
- `identity`

  The identity method identifies features in the input layer with features in the method layer possibly splitting features into several features. By default the result layer has attributes from both operand layers.



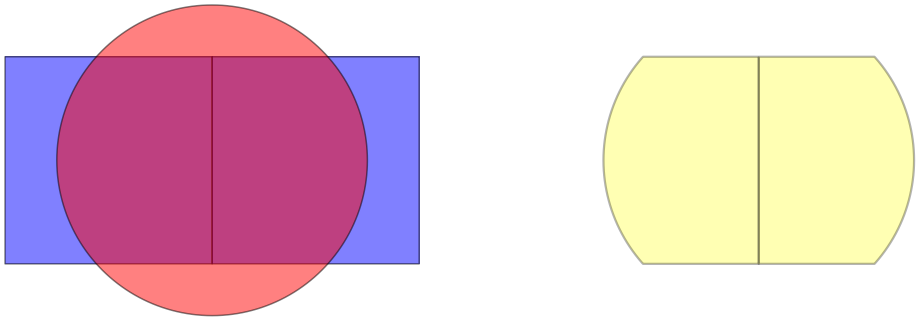
- `update`

  The update method creates a layer, which add features into the input layer from the method layer possibly cutting features in the input layer. By default the result layer has attributes only from the input layer.



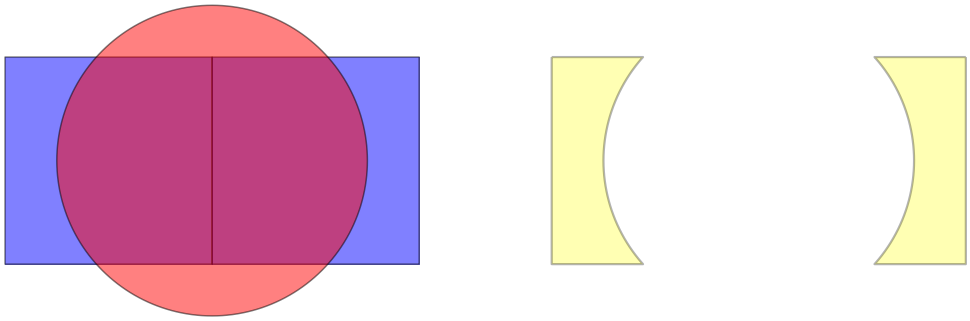
- `clip`

  The clip method creates a layer, which has features from the input layer clipped to the areas of the features in the method layer. By default the result layer has attributes of the input layer.



- `erase`

  The erase method creates a layer, which has features from the input layer whose areas are erased by the features in the method layer. By default the result layer has attributes of the input layer.



--method-layer <METHOD-LAYER>

Name of the method vector layer.

Advanced options

--all-input-field

Add all input fields to output layer. Mutually exclusive with *--input-field*, *--no-input-field*.

--all-method-field

Add all method fields to output layer. Mutually exclusive with *--method-field*, *--no-method-field*.

--input-field <INPUT-FIELD>

Input field(s) to add to output layer [may be repeated] Mutually exclusive with *--no-input-field*, *--all-input-field*.

--input-prefix <INPUT-PREFIX>

Prefix for fields corresponding to input layer. Defaults to `input_` if there are both input and method fields, otherwise empty string.

--method-field <METHOD-FIELD>

Input field(s) to add to output layer [may be repeated] Mutually exclusive with *--no-method-field*, *--all-method-field*.

--method-prefix <METHOD-PREFIX>

Prefix for fields corresponding to method layer. Defaults to `method_` if there are both input and method fields, otherwise empty string.

--no-input-field

Do not add any input field to output layer. Mutually exclusive with *--input-field*, *--all-input-field*.

--no-method-field

Do not add any method field to output layer. Mutually exclusive with *--method-field*, *--all-method-field*.

Standard Options

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-layer <NAME>

Specifies the name of the layer to which features will be written. If not specified, output layer names will be the same as input layer names.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--update

Whether to open an existing output dataset in update mode.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Performs a union between both input and method layers.

```
$ gdal vector layer-algebra union input.shp method.shp output.shp
```

5.1.4.22 gdal vector make-point

Added in version 3.12.

Synopsis

```
Usage: gdal vector make-point [OPTIONS] <INPUT> <OUTPUT>

Create point geometries from attribute fields

Positional arguments:
  -i, --input <INPUT>                                    Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>               Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>           Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                               Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                      Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                               Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                         Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                          Output layer name [not available␣
↪in pipelines]
  --skip-errors                                          Skip errors when writing features␣
↪[not available in pipelines]
```

(continues on next page)

(continued from previous page)

```
  --x <X>                                                   Field from which X coordinate␣
↪should be read [required]
  --y <Y>                                                   Field from which Y coordinate␣
↪should be read [required]
  --z <Z>                                                   Optional field from which Z␣
↪coordinate should be read
  --m <M>                                                   Optional field from which M␣
↪coordinate should be read
  --output-crs <OUTPUT-CRS>                                 Output CRS

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                       Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                         Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>           Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                                  Upsert features (implies 'append
↪') [not available in pipelines]
                                                            Mutually exclusive with --append
```

Description

gdal vector make-point creates point geometries from attribute fields containing coordinates.

Program-Specific Options

--output-crs <CRS>

Optional coordinate reference system to assign to the created points. The coordinate reference systems that can be passed are anything supported by the *OGRSpatialReference::SetFromUserInput()* call, which includes EPSG Projected, Geographic or Compound CRS (i.e. EPSG:4296), a well known text (WKT) CRS definition, PROJ.4 declarations, or the name of a .prj file containing a WKT CRS definition.

--m <FIELD_NAME>

Optional field name containing M coordinate values

--x <FIELD_NAME>

Field name containing X coordinate values

--y <FIELD_NAME>

Field name containing Y coordinate values

--z <FIELD_NAME>

Optional field name containing Z coordinate values

Standard Options

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--input-layer <INPUT-LAYER>

Specifies the name of one or more layers to process. By default, all layers will be processed.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-layer <NAME>

Specifies the name of the layer to which features will be written. If not specified, output layer names will be the same as input layer names.

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

5.1.4.23 gdal vector make-valid

Added in version 3.12.

Synopsis

```
Usage: gdal vector make-valid [OPTIONS] <INPUT> <OUTPUT>

Fix validity of geometries of a vector dataset.

Positional arguments:
  -i, --input <INPUT>                                    Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
```

(continues on next page)

(continued from previous page)

```
↪repeated]
  -q, --quiet                                          Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>             Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>         Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                          Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                             Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                    Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                             Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                       Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                        Output layer name [not available␣
↪in pipelines]
  --skip-errors                                        Skip errors when writing features␣
↪[not available in pipelines]
  --active-layer <ACTIVE-LAYER>                        Set active layer (if not␣
↪specified, all)
  --active-geometry <ACTIVE-GEOMETRY>                  Geometry field name to which to␣
↪restrict the processing (if not specified, all)
  --method <METHOD>                                    Algorithm to use when repairing␣
↪invalid geometries.. METHOD=linework|structure (default: linework)
  --keep-lower-dim                                     Keep components of lower␣
↪dimension after MakeValid()

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                  Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                    Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>      Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                             Upsert features (implies 'append
↪') [not available in pipelines]
                                                       Mutually exclusive with --append
```

Description

gdal vector make-valid ensures that geometries are valid regarding the rules of the Simple Features specification.

It runs the *OGRGeometry::MakeValid()* operation, followed by *OGRGeometryFactory::removeLowerDimensionSubGeoms()* (unless *--keep-lower-dim* is set)

It can also be used as a step of *gdal vector pipeline*.

> **Note**
>
> This command requires a GDAL build against the GEOS library.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

--keep-lower-dim

Keep components of lower dimension after MakeValid(). For example, MakeValid() may return a GeometryCollection of a Polygon and a LineString from a Polygon input. By default only the Polygon would be returned. Setting this option will return the GeometryCollection.

--method=`linework|structure`

Algorithm to use when repairing invalid geometries.

The default `linework` method combines all rings into a set of noded lines and then extracts valid polygons from that linework. This method keeps all input vertices.

The `structure` method (only available with GEOS >= 3.10) first makes all rings valid then merges shells and subtracts holes from shells to generate valid result. It assumes that holes and shells are correctly categorized.

For examples of the differences between these methods, see *Geometry Validity*.

Standard Options

--active-geometry `<ACTIVE-GEOMETRY>`

Set the active geometry field from its name. When it is specified, only the specified geometry field will be subject to the processing. Other geometry fields will be not modified. If this option is not specified, all geometry fields will be subject to the processing. This option can be combined together with `--active-layer`.

--active-layer `<ACTIVE-LAYER>`

Set the active layer. When it is specified, only the layer specified by its name will be subject to the processing. Other layers will be not modified. If this option is not specified, all layers will be subject to the processing.

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option `<NAME>=<VALUE>`

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--input-layer <INPUT-LAYER>

Specifies the name of one or more layers to process. By default, all layers will be processed. To read and write all layers but only process a subset, use *--active-layer*.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-layer <NAME>

Specifies the name of the layer to which features will be written. If not specified, output layer names will be the same as input layer names.

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Basic use of make-valid

```
$ gdal vector make-valid in.gpkg out.gpkg --overwrite
```

5.1.4.24 gdal vector materialize

Added in version 3.12.

Description

The `materialize` operation is for use only in a pipeline, and writes the incoming dataset to a dataset on disk, before passing that materialized dataset to following steps. This can be used when materializing a dataset helps for performance compared to on-demand evaluation of previous steps.

By default, a GeoPackage file (or a Spatialite file if GeoPackage cannot support some features of the input dataset) is used for the materialized dataset, created in the directory pointed by the *CPL_TMPDIR* configuration option or in the current directory if not specified, and is deleted when the pipeline finishes. The user can also select another format, or specify an explicit filename, in which case the materialized dataset is not deleted when the pipeline finishes.

Synopsis

```
* materialize [OPTIONS]
------------------------

Materialize a piped dataset on disk to increase the efficiency of the following steps.

Options:
```

(continues on next page)

(continued from previous page)

```
  -o, --output <OUTPUT>                                  Materialized dataset name␣
↪(created by algorithm)
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]
  --lco, --layer-creation-option <KEY>=<VALUE>           Layer creation option [may be␣
↪repeated]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed
```

Program-Specific Options

--output <OUTPUT>

Optional dataset name. When specified, it is not removed at the end of the process.

Standard Options

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Reproject a GeoPackage file to CRS EPSG:32632 ("WGS 84 / UTM zone 32N"), materialize it to a temporary file and rasterize it

```
$ gdal pipeline ! read in.gpkg ! reproject --output-crs=EPSG:32632 ! \
                ! materialize ! rasterize --resolution 10,10 ! write out.gpkg --overwrite
```

5.1.4.25 gdal vector partition

Added in version 3.12.

Synopsis

```
Usage: gdal vector partition [OPTIONS] <INPUT> <OUTPUT>

Partition a vector dataset into multiple files.

Positional arguments:
  -i, --input <INPUT>                                    Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output directory [required]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed
                                                         Mutually exclusive with --append
  --append                                               Whether appending to existing␣
↪layer is allowed
                                                         Mutually exclusive with --
↪overwrite
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]
  --lco, --layer-creation-option <KEY>=<VALUE>           Layer creation option [may be␣
↪repeated]
  --field <FIELD>                                        Attribute or geometry field(s) on␣
↪which to partition [may be repeated]
  --scheme <SCHEME>                                      Partitioning scheme.␣
↪SCHEME=hive|flat (default: hive)
  --pattern <PATTERN>                                    Filename pattern ('part_%010d'␣
↪for scheme=hive, '{LAYER_NAME}_{FIELD_VALUE}_%010d' for scheme=flat)
  --feature-limit <FEATURE-LIMIT>                        Maximum number of features per␣
↪file
  --max-file-size <MAX-FILE-SIZE>                        Maximum file size (MB or GB␣
```

(continues on next page)

(continued from previous page)

```
↪suffix can be used)
  --omit-partitioned-field                          Whether to omit partitioned␣
↪fields from target layer definition
  --skip-errors                                     Skip errors when writing features

Advanced Options:
  --if, --input-format <INPUT-FORMAT>               Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                 Open options [may be repeated]␣
↪[not available in pipelines]
```

Description

gdal vector partition dispatches features into different files, depending on the values the feature take on a subset of attribute or geometry fields specified by the user and/or by limiting each output layer to a maximum number of features and/or a maximum file size.

Two partitioning schemes are available:

- `hive`, corresponding to Apache Hive partitioning, is the default one.

  Each partitioning field corresponds to a nested directory. Let's consider a layer with fields "continent" and "country", chosen as partitioning fields. All features where "continent" evaluates to "Europe" and "country" evaluates to "France", will be written in the "continent=Europe/country=France/" subdirectory of the output directory.

  NULL values for partitioning fields are encoded as `__HIVE_DEFAULT_PARTITION__` in the directory name. Non-ASCII characters, space, equal sign, or characters not compatible with directory name constraints are percent-encoded (e.g. `%20` for space).

- `flat` where files are written directly under the output directory using a default filename pattern of `{LAYER_NAME}_{FIELD_VALUE}_%10d`.

By default, the format of the input dataset will be used for the output, if it can be determined and the input driver supports writing. Otherwise, *--format* must be used.

gdal vector partition can be used as the last step of a pipeline.

The following options are available:

Program-Specific Options

--feature-limit <FEATURE-LIMIT>

Maximum number of features per file. By default, unlimited. If the limit is exceeded, several parts are created.

--field <FIELD-NAME>

Fields(s) on which to partition

Only attribute fields of type String, Integer and Integer64 are allowed. The order into which fields are specified matter to determine the directory hierarchy.

Starting with GDAL 3.13, geometry field names can be specified (`OGR_GEOMETRY` being the generic name for the first geometry field). Partitioning on geometry fields is done on the geometry type. This can be useful for file formats where a single geometry type per layer is allowed.

Starting with GDAL 3.13, *--field* is no longer required, but when it is not specified, *--feature-limit* and/or *--max-file-size* must be specified.

--max-file-size <MAX-FILE-SIZE>

Maximum file size (MB or GB suffix can be used). By default, unlimited. If the limit is exceeded, several parts are created.

Note that the maximum file size is used as a hint, and might not be strictly respected, because the evaluation of the file size corresponding to a feature is based on a heuristics, as the file size itself cannot be reliably used when it is under writing. In particular, the heuristics does not assume any compression, so for compressed formats, the actual size of a part can be significantly smaller than the specified limit.

--omit-partitioned-field

Whether to omit partitioned fields from the target layer definition. Automatically set for Parquet output format and Hive partitioning.

--output <OUTPUT-DIRECTORY>

Root of the output directory. [required]

--pattern <PATTERN>

Filename pattern. User chosen string, with substitutions for:

- `{LAYER_NAME}`, when found, is substituted with the layer name (percent encoded where needed).
- `{FIELD_VALUE}`, when found, is substituted with the partitioning field value (percent encoded where needed). If several partitioning fields are used, each value is separated by underscore (_). Empty strings are substituted with `__EMPTY__` and null fields with `__NULL__`.
- `%[0?][0-9]?[0]?d`: C-style integer formatter for the part number. Valid values are for example `%d` or `%05d`. One and only one part number specifier must be present in the pattern.

Default values for the pattern are `part_%010d` for the hive scheme, and `{LAYER_NAME}_{FIELD_VALUE}_%010d` for the flat scheme.`

--scheme `hive|flat`

Partitioning scheme. Defaults to `hive`.

Standard Options

--append

Whether the output directory must be opened in append mode. Implies that it already exists and that the output format supports appending.

This mode is useful when adding new features to an already an existing partitioned dataset.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Create a partition based on the "continent" and "country" fields

```
$ gdal vector partition world_cities.gpkg out_directory --field continent,country --
↪format Parquet
```

Example 2: Create a partition based on the "country" field, filtering on cities with population bigger than 1 million, with a flat partitioning scheme

```
$ gdal pipeline ! read world_cities.gpkg ! filter --where "pop > 1e6" ! partition out_
↪directory --field country --format GPKG --scheme flat
```

Example 3: Create multiple Shapefiles, one for each geometry type, from a GeoPackage file, by grouping together POLYGON/MULTIPOLYGON or LINESTRING/MULTILINESTRING.

```
$ gdal vector pipeline ! read input.gpkg ! set-geom-type --multi ! partition out_
↪directory --scheme flat --field OGR_GEOMETRY --format "ESRI Shapefile"
```

Example 4: Split a file into files with at most 100,000 features.

```
$ gdal vector partition world_cities.gpkg out_directory --feature-limit 100000 --scheme␣
↪flat --format Parquet
```

Example 5: Sort a file spatially and split it into files with at most 100,000 features.

```
$ gdal vector pipeline read world_cities.gpkg ! sort ! partition out_directory --feature-
↪limit 100000 --scheme flat --format Parquet
```

5.1.4.26 gdal vector pipeline

Added in version 3.11.

Description

gdal vector pipeline can be used to process a vector dataset and perform various processing steps that accept vector and generate vector.

For pipelines mixing raster and vector, consult *gdal pipeline*.

Most steps proceed in on-demand evaluation of features, unless otherwise stated in their documentation, without "materializing" the resulting dataset of the operation of each step. It may be desirable sometimes for performance purposes to proceed to materializing an intermediate dataset to disk using *gdal vector materialize*.

Synopsis

```
Usage: gdal vector pipeline [OPTIONS] <PIPELINE>

Process a vector dataset applying several steps.

Positional arguments:

Common Options:
  -h, --help              Display help message and exit
  --json-usage            Display usage as JSON document and exit
  --config <KEY>=<VALUE>  Configuration option [may be repeated]
  -q, --quiet             Quiet mode (no progress bar or warning message) [not available␣
↪in pipelines]

Options:
  --skip-errors           Skip errors when writing features [not available in pipelines]

<PIPELINE> is of the form: read|concat [READ-OPTIONS] ( ! <STEP-NAME> [STEP-OPTIONS] )* !
↪ write|info [WRITE-OPTIONS]
```

A pipeline chains several steps, separated with the *!* (exclamation mark) character. The first step must be `read` or `concat`, and the last one `export-schema`, `info`, `partition` or `write`. Each step has its own positional or non-positional arguments. Apart from `read`, `concat`, `info`, `partition` and `write`, all other steps can potentially be used several times in a pipeline.

Steps

- *buffer*
- *check-coverage*
- *check-geometry*
- *clean-coverage*
- *clip*
- *combine*
- *concave-hull*
- *concat*
- *convex-hull*
- *create*
- *dissolve*
- *edit*
- *explode-collections*
- *export-schema*
- *external*
- *filter*
- *info*
- *limit*
- *make-point*
- *make-valid*
- *materialize*
- *partition*
- *read*
- *rename-layer*
- *reproject*
- *segmentize*
- *select*
- *set-field-type*
- *set-geom-type*
- *simplify*

- *simplify-coverage*
- *sort*
- *sql*
- *swap-xy*
- *tee*
- *update*
- *write*

buffer

```
* buffer [OPTIONS] <DISTANCE>
----------------------------

Compute a buffer around geometries of a vector dataset.

Positional arguments:
  --distance <DISTANCE>                    Distance to which to extend the geometry.␣
↪[required]

Options:
  --active-layer <ACTIVE-LAYER>            Set active layer (if not specified, all)
  --active-geometry <ACTIVE-GEOMETRY>      Geometry field name to which to restrict the␣
↪processing (if not specified, all)
  --endcap-style <ENDCAP-STYLE>            Endcap style.. ENDCAP-STYLE=round|flat|square␣
↪(default: round)
  --join-style <JOIN-STYLE>                Join style.. JOIN-STYLE=round|mitre|bevel␣
↪(default: round)
  --mitre-limit <MITRE-LIMIT>              Mitre ratio limit (only affects mitered join␣
↪style). (default: 5)
  --quadrant-segments <QUADRANT-SEGMENTS>  Number of line segments used to approximate a␣
↪quarter circle. (default: 8)
  --side <SIDE>                            Sets whether the computed buffer should be␣
↪single-sided or not.. SIDE=both|left|right (default: both)
```

Details for options can be found in *gdal vector buffer*.

check-coverage

```
* check-coverage [OPTIONS]
--------------------------

Check a polygon coverage for validity

Options:
  --include-valid                          Include valid inputs in output, with empty␣
↪geometry
  --geometry-field <GEOMETRY-FIELD>        Name of geometry field to check
  --maximum-gap-width <MAXIMUM-GAP-WIDTH>  Maximum width of a gap to be flagged
```

Details for options can be found in *gdal vector check-coverage*.

check-geometry

```
* check-geometry [OPTIONS]
--------------------------

Check a dataset for invalid geometries

Options:
  --include-field <INCLUDE-FIELD>    Fields from input layer to include in output␣
↪(special values: ALL and NONE) (default: NONE) [may be repeated]
  --include-valid                    Include valid inputs in output, with empty geometry
  --geometry-field <GEOMETRY-FIELD>  Name of geometry field to check
```

Details for options can be found in *gdal vector check-geometry*.

clean-coverage

```
* clean-coverage [OPTIONS]
--------------------------

Alter polygon boundaries to make shared edges identical, removing gaps and overlaps

Options:
  --active-layer <ACTIVE-LAYER>            Set active layer (if not specified, all)
  --snapping-distance <SNAPPING-DISTANCE>  Distance tolerance for snapping nodes
  --merge-strategy <MERGE-STRATEGY>        Algorithm to assign overlaps to neighboring␣
↪polygons. MERGE-STRATEGY=longest-border|max-area|min-area|min-index
  --maximum-gap-width <MAXIMUM-GAP-WIDTH>  Maximum width of a gap to be closed
```

Details for options can be found in *gdal vector clean-coverage*.

clip

```
* clip [OPTIONS]
----------------

Clip a vector dataset.

Options:
  --active-layer <ACTIVE-LAYER>    Set active layer (if not specified, all)
  --bbox <BBOX>                    Clipping bounding box as xmin,ymin,xmax,ymax
                                   Mutually exclusive with --geometry, --like
  --bbox-crs <BBOX-CRS>            CRS of clipping bounding box
  --geometry <GEOMETRY>            Clipping geometry (WKT or GeoJSON)
                                   Mutually exclusive with --bbox, --like
  --geometry-crs <GEOMETRY-CRS>    CRS of clipping geometry
  --like <DATASET>                 Dataset to use as a template for bounds
                                   Mutually exclusive with --bbox, --geometry
  --like-sql <SELECT-STATEMENT>    SELECT statement to run on the 'like' dataset
                                   Mutually exclusive with --like-where
```

(continues on next page)

(continued from previous page)

```
  --like-layer <LAYER-NAME>        Name of the layer of the 'like' dataset
  --like-where <WHERE-EXPRESSION>  WHERE SQL clause to run on the 'like' dataset
                                   Mutually exclusive with --like-sql
```

Details for options can be found in *gdal vector clip*.

combine

```
* combine [OPTIONS]
-------------------

Combine features into collections

Options:
  --group-by <GROUP-BY>                  Names of field(s) by which inputs should be␣
↪grouped [may be repeated]
  --keep-nested                          Avoid combining the components of multipart␣
↪geometries
  --add-extra-fields <ADD-EXTRA-FIELDS>  Whether to add extra fields, depending on if␣
↪they have identical values within each group. ADD-EXTRA-FIELDS=no|sometimes-
↪identical|always-identical (default: no)
```

Details for options can be found in *gdal vector combine*.

concave-hull

```
* concave-hull [OPTIONS]
------------------------

Compute the concave hull of geometries of a vector dataset.

Options:
  --active-layer <ACTIVE-LAYER>        Set active layer (if not specified, all)
  --active-geometry <ACTIVE-GEOMETRY>  Geometry field name to which to restrict the␣
↪processing (if not specified, all)
  --ratio <RATIO>                      Ratio controlling the concavity [required]
  --allow-holes                        Allow holes in the output polygon
  --tight                              Whether the hull must follow the outer boundaries␣
↪of the input polygons
```

Details for options can be found in *gdal vector concave-hull*.

concat

```
* concat [OPTIONS] <INPUTS>...
------------------------------

Concatenate vector datasets.

Positional arguments:
  -i, --input <INPUTS>                                        Input vector datasets [1..␣
```

(continues on next page)

(continued from previous page)

```
↪values] [required]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>                  Input layer name(s) [may be␣
↪repeated]
  --mode <MODE>                                             Determine the strategy to␣
↪create output layers from source layers . MODE=merge-per-layer-name|stack|single␣
↪(default: merge-per-layer-name)
  --output-layer <OUTPUT-LAYER>                             Name of the output vector␣
↪layer (single mode), or template to name the output vector layers (stack mode)
  --source-layer-field-name <SOURCE-LAYER-FIELD-NAME>       Name of the new field to␣
↪add to contain identification of the source layer, with value determined from 'source-
↪layer-field-content'
  --source-layer-field-content <SOURCE-LAYER-FIELD-CONTENT>  A string, possibly using
↪{AUTO_NAME}, {DS_NAME}, {DS_BASENAME}, {DS_INDEX}, {LAYER_NAME}, {LAYER_INDEX}
  --field-strategy <FIELD-STRATEGY>                         How to determine target␣
↪fields from source fields. FIELD-STRATEGY=union|intersection (default: union)
  -s, --input-crs <INPUT-CRS>                               Input CRS
  -d, --output-crs <OUTPUT-CRS>                             Output CRS

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                       Input formats [may be␣
↪repeated]
  --oo, --open-option <KEY>=<VALUE>                         Open options [may be␣
↪repeated]
```

Details for options can be found in *gdal vector concat*.

convex-hull

```
* convex-hull [OPTIONS]
-----------------------

Compute the convex hull of geometries of a vector dataset.

Options:
  --active-layer <ACTIVE-LAYER>        Set active layer (if not specified, all)
  --active-geometry <ACTIVE-GEOMETRY>  Geometry field name to which to restrict the␣
↪processing (if not specified, all)
```

Details for options can be found in *gdal vector convex-hull*.

create

```
* create [OPTIONS] <OUTPUT>
---------------------------

Create a vector dataset.

Positional arguments:
  -o, --output <OUTPUT>                                Output vector dataset [required]
```

(continues on next page)

(continued from previous page)

```
Options:
  -i, --like, --input <TEMPLATE-DATASET>                Input vector datasets
                                                        Mutually exclusive with --schema,␣
↪--field
  -l, --layer, --input-layer <INPUT-LAYER>              Input layer name(s) [may be␣
↪repeated]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)
  --co, --creation-option <KEY>=<VALUE>                 Creation option [may be repeated]
  --lco, --layer-creation-option <KEY>=<VALUE>          Layer creation option [may be␣
↪repeated]
  --overwrite                                           Whether overwriting existing␣
↪output dataset is allowed
  --update                                              Whether to open existing dataset␣
↪in update mode
  --overwrite-layer                                     Whether overwriting existing␣
↪output layer is allowed
  --output-layer <OUTPUT-LAYER>                         Output layer name
  --geometry-type <GEOMETRY-TYPE>                       Layer geometry type
  --geometry-field <GEOMETRY-FIELD>                     Name of the geometry field to␣
↪create (if supported by the output format) (default: geom)
  --crs <CRS>                                           Set CRS
  --fid <FID>                                           FID column name
  --schema <SCHEMA_JSON>                                Read OGR_SCHEMA and populate␣
↪field definitions from it
                                                        Mutually exclusive with --input, -
↪-field
  --field <NAME>:<TYPE>[(,<WIDTH>[,<PRECISION>])]>      Add a field definition to the␣
↪output layer [may be repeated]
                                                        Mutually exclusive with --input, -
↪-schema

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                   Input formats [may be repeated]
  --oo, --open-option <KEY>=<VALUE>                     Open options [may be repeated]
  --output-oo, --output-open-option <KEY>=<VALUE>       Output open options [may be␣
↪repeated]
```

Details for options can be found in *gdal vector create*.

dissolve

```
* dissolve [OPTIONS]
--------------------

Dissolves multipart features

Options:
  --active-layer <ACTIVE-LAYER>        Set active layer (if not specified, all)
  --active-geometry <ACTIVE-GEOMETRY>  Geometry field name to which to restrict the␣
↪processing (if not specified, all)
```

Details for options can be found in *gdal vector dissolve*.

edit

```
* edit [OPTIONS]
----------------

Edit metadata of a vector dataset.

Options:
  --active-layer <ACTIVE-LAYER>    Set active layer (if not specified, all)
  --output-layer <OUTPUT-LAYER>    Output layer name
  --geometry-type <GEOMETRY-TYPE>  Layer geometry type
  --crs <CRS>                      Override CRS (without reprojection)
  --metadata <KEY>=<VALUE>         Add/update dataset metadata item [may be repeated]
  --unset-metadata <KEY>           Remove dataset metadata item [may be repeated]
  --layer-metadata <KEY>=<VALUE>   Add/update layer metadata item [may be repeated]
  --unset-layer-metadata <KEY>     Remove layer metadata item [may be repeated]
  --unset-fid                      Unset the identifier of each feature and the FID␣
↪column name
```

Details for options can be found in *gdal vector edit*.

explode-collections

```
* explode-collections [OPTIONS]
-------------------------------

Explode geometries of type collection of a vector dataset.

Options:
  --active-layer <ACTIVE-LAYER>        Set active layer (if not specified, all)
  --active-geometry <ACTIVE-GEOMETRY>  Geometry field name to which to restrict the␣
↪processing (if not specified, all)
  --geometry-type <GEOMETRY-TYPE>      Geometry type
  --skip-on-type-mismatch              Skip feature when change of feature geometry type␣
↪failed
```

Details for options can be found in *gdal vector explode-collections*.

export-schema

Added in version 3.12.

```
* export-schema [OPTIONS]
-------------------------

Export the OGR_SCHEMA from a vector dataset.

Positional arguments:

Options:
  -l, --layer, --input-layer <INPUT-LAYER>  Input layer name [may be repeated]
```

(continues on next page)

(continued from previous page)

```
  -o, --output <OUTPUT>                     Output file name. If not specified, output␣
↪is sent to stdout
  --overwrite                               Whether overwriting existing output file is␣
↪allowed
```

Details for options can be found in *gdal vector export-schema*.

external

```
* external [OPTIONS] <COMMAND>
------------------------------

Execute an external program as a step of a pipeline

Positional arguments:
  --command <COMMAND>                                  External command, optionally with
↪<INPUT> and/or <OUTPUT> or <INPUT-OUTPUT> placeholders [required]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                  Input formats
```

Details for options can be found in *gdal external*.

filter

```
* filter [OPTIONS]
------------------

Filter a vector dataset.

Options:
  --active-layer <ACTIVE-LAYER>  Set active layer (if not specified, all)
  --bbox <BBOX>                  Bounding box as xmin,ymin,xmax,ymax
  --where <WHERE>|@<filename>    Attribute query in a restricted form of the queries␣
↪used in the SQL WHERE statement
  --update-extent                Update layer extent to take into account the filter
```

Details for options can be found in *gdal vector filter*.

info

Added in version 3.12.

```
* info [OPTIONS]
----------------

Return information on a vector dataset.

Positional arguments:
```

(continues on next page)

(continued from previous page)

```
Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format. OUTPUT-
↪FORMAT=json|text
  -l, --layer, --input-layer <INPUT-LAYER>             Input layer name [may be repeated]
                                                       Mutually exclusive with --sql, --
↪fid
  --features                                           List all features (beware of RAM␣
↪consumption on large layers)
                                                       Mutually exclusive with --summary
  --summary                                            List the layer names and the␣
↪geometry type
                                                       Mutually exclusive with --features
  --limit <FEATURE-COUNT>                              Limit the number of features per␣
↪layer (implies --features)
  --sql <statement>|@<filename>                        Execute the indicated SQL␣
↪statement and return the result
                                                       Mutually exclusive with --input-
↪layer, --fid
  --where <WHERE>|@<filename>                          Attribute query in a restricted␣
↪form of the queries used in the SQL WHERE statement
  --fid <FID>                                          Feature identifier
                                                       Mutually exclusive with --input-
↪layer, --sql
  --dialect <DIALECT>                                  SQL dialect

Esoteric Options:
  --crs-format <CRS-FORMAT>                            Which format to use to report CRS.
↪ CRS-FORMAT=AUTO|WKT2|PROJJSON (default: AUTO)
```

Details for options can be found in *gdal vector info*.

limit

```
* limit [OPTIONS] <LIMIT>
-------------------------

Truncate a vector dataset to no more than a specified number of features.

Positional arguments:
  --limit <LIMIT>                Limit the number of features to read per layer␣
↪[required]

Options:
  --active-layer <ACTIVE-LAYER>  Set active layer (if not specified, all)
```

make-point

```
* make-point [OPTIONS]
----------------------
```

(continues on next page)

(continued from previous page)

```
Create point geometries from attribute fields

Options:
  --x <X>                    Field from which X coordinate should be read [required]
  --y <Y>                    Field from which Y coordinate should be read [required]
  --z <Z>                    Optional field from which Z coordinate should be read
  --m <M>                    Optional field from which M coordinate should be read
  --output-crs <OUTPUT-CRS>  Output CRS
```

Details for options can be found in *gdal vector make-point*.

make-valid

```
* make-valid [OPTIONS]
----------------------

Fix validity of geometries of a vector dataset.

Options:
  --active-layer <ACTIVE-LAYER>        Set active layer (if not specified, all)
  --active-geometry <ACTIVE-GEOMETRY>  Geometry field name to which to restrict the␣
↪processing (if not specified, all)
  --method <METHOD>                    Algorithm to use when repairing invalid␣
↪geometries.. METHOD=linework|structure (default: linework)
  --keep-lower-dim                     Keep components of lower dimension after␣
↪MakeValid()
```

Details for options can be found in *gdal vector make-valid*.

materialize

```
* materialize [OPTIONS]
-----------------------

Materialize a piped dataset on disk to increase the efficiency of the following steps.

Options:
  -o, --output <OUTPUT>                                Materialized dataset name␣
↪(created by algorithm)
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format
  --co, --creation-option <KEY>=<VALUE>                Creation option [may be repeated]
  --lco, --layer-creation-option <KEY>=<VALUE>         Layer creation option [may be␣
↪repeated]
  --overwrite                                          Whether overwriting existing␣
↪output dataset is allowed
```

Details for options can be found in *gdal vector materialize*.

partition

Added in version 3.12.

```
* partition [OPTIONS] <OUTPUT>
------------------------------

Partition a vector dataset into multiple files.

Positional arguments:
  -o, --output <OUTPUT>                                Output directory [required]

Options:
  --overwrite                                          Whether overwriting existing␣
↪output dataset is allowed
                                                       Mutually exclusive with --append
  --append                                             Whether appending to existing␣
↪layer is allowed
                                                       Mutually exclusive with --
↪overwrite
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format
  --co, --creation-option <KEY>=<VALUE>                Creation option [may be repeated]
  --lco, --layer-creation-option <KEY>=<VALUE>         Layer creation option [may be␣
↪repeated]
  --field <FIELD>                                      Attribute or geometry field(s) on␣
↪which to partition [may be repeated]
  --scheme <SCHEME>                                    Partitioning scheme.␣
↪SCHEME=hive|flat (default: hive)
  --pattern <PATTERN>                                  Filename pattern ('part_%010d'␣
↪for scheme=hive, '{LAYER_NAME}_{FIELD_VALUE}_%010d' for scheme=flat)
  --feature-limit <FEATURE-LIMIT>                      Maximum number of features per␣
↪file
  --max-file-size <MAX-FILE-SIZE>                      Maximum file size (MB or GB␣
↪suffix can be used)
  --omit-partitioned-field                             Whether to omit partitioned␣
↪fields from target layer definition
  --skip-errors                                        Skip errors when writing features
```

Details for options can be found in *gdal vector partition*.

read

```
* read [OPTIONS] <INPUT>
------------------------

Read a vector dataset.

Positional arguments:
  -i, --input <INPUT>                       Input vector datasets [required]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>  Input layer name(s) [may be repeated]
```

(continues on next page)

(continued from previous page)

```
Advanced Options:
  --if, --input-format <INPUT-FORMAT>       Input formats [may be repeated]
  --oo, --open-option <KEY>=<VALUE>         Open options [may be repeated]
```

Details for options can be found in *gdal vector pipeline read*.

rename-layer

```
* rename-layer [OPTIONS]
------------------------

Rename layer(s) of a vector dataset.

Options:
  -l, --input-layer <INPUT-LAYER>                  Input layer name
  --output-layer <OUTPUT-LAYER>                    Output layer name
  --ascii                                          Force names to ASCII character
  --lower-case                                     Force names to lower case (only on␣
↪ASCII characters)
  --filename-compatible                            Force names to be usable as filenames
  --reserved-characters <RESERVED-CHARACTERS>      Reserved character(s) to be removed
  --replacement-character <REPLACEMENT-CHARACTER>  Replacement character when ASCII␣
↪conversion not possible
  --max-length <MAX-LENGTH>                        Maximum length of layer names
```

Details for options can be found in *gdal vector rename-layer*.

reproject

```
* reproject [OPTIONS]
---------------------

Reproject a vector dataset.

Options:
  --active-layer <ACTIVE-LAYER>  Set active layer (if not specified, all)
  -s, --input-crs <INPUT-CRS>    Input CRS
  -d, --output-crs <OUTPUT-CRS>  Output CRS [required]
```

Details for options can be found in *gdal vector reproject*.

segmentize

```
* segmentize [OPTIONS] <MAX-LENGTH>
-----------------------------------

Segmentize geometries of a vector dataset.

Positional arguments:
  --max-length <MAX-LENGTH>            Maximum length of a segment [required]
```

(continues on next page)

(continued from previous page)

```
Options:
  --active-layer <ACTIVE-LAYER>        Set active layer (if not specified, all)
  --active-geometry <ACTIVE-GEOMETRY>  Geometry field name to which to restrict the␣
↪processing (if not specified, all)
```

Details for options can be found in *gdal vector segmentize*.

select

```
* select [OPTIONS] <FIELDS>
---------------------------

Select a subset of fields from a vector dataset.

Positional arguments:
  --fields <FIELDS>              Fields to select (or exclude if --exclude) [may be␣
↪repeated] [required]

Options:
  --active-layer <ACTIVE-LAYER>  Set active layer (if not specified, all)
  --output-layer <OUTPUT-LAYER>  Output layer name
  --exclude                      Exclude specified fields
                                 Mutually exclusive with --ignore-missing-fields
  --ignore-missing-fields        Ignore missing fields
                                 Mutually exclusive with --exclude
```

Details for options can be found in *gdal vector select*.

set-field-type

```
* set-field-type [OPTIONS]
--------------------------

Modify the type of a field of a vector dataset.

Options:
  --active-layer <ACTIVE-LAYER>                   Set active layer (if not specified,␣
↪all)
  --field-name <FIELD-NAME>                       Field name [required]
                                                  Mutually exclusive with --input-field-
↪type
  --input-field-type <INPUT-FIELD-TYPE>           Source field type or subtype [required]
                                                  Mutually exclusive with --field-name
  --output-field-type, --field-type <FIELD-TYPE>  Target field type or subtype [required]
```

Details for options can be found in *gdal vector set-field-type*.

set-geom-type

```
* set-geom-type [OPTIONS]
-------------------------
```

(continues on next page)

(continued from previous page)

```
Modify the geometry type of a vector dataset.

Options:
  --active-layer <ACTIVE-LAYER>        Set active layer (if not specified, all)
  --active-geometry <ACTIVE-GEOMETRY>  Geometry field name to which to restrict the␣
↪processing (if not specified, all)
  --layer-only                         Only modify the layer geometry type
                                       Mutually exclusive with --feature-only
  --feature-only                       Only modify the geometry type of features
                                       Mutually exclusive with --layer-only
  --geometry-type <GEOMETRY-TYPE>      Geometry type
  --multi                              Force geometries to MULTI geometry types
                                       Mutually exclusive with --single
  --single                             Force geometries to non-MULTI geometry types
                                       Mutually exclusive with --multi
  --linear                             Convert curve geometries to linear types
                                       Mutually exclusive with --curve
  --curve                              Convert linear geometries to curve types
                                       Mutually exclusive with --linear
  --dim <DIM>                          Force geometries to the specified dimension.␣
↪DIM=XY|XYZ|XYM|XYZM
  --skip                               Skip feature when change of feature geometry type␣
↪failed
```

Details for options can be found in *gdal vector set-geom-type*.

simplify

```
* simplify [OPTIONS] <TOLERANCE>
--------------------------------

Simplify geometries of a vector dataset.

Positional arguments:
  --tolerance <TOLERANCE>              Distance tolerance for simplification. [required]

Options:
  --active-layer <ACTIVE-LAYER>        Set active layer (if not specified, all)
  --active-geometry <ACTIVE-GEOMETRY>  Geometry field name to which to restrict the␣
↪processing (if not specified, all)
```

Details for options can be found in *gdal vector simplify*.

simplify-coverage

```
* simplify-coverage [OPTIONS] <TOLERANCE>
-----------------------------------------

Simplify shared boundaries of a polygonal vector dataset.
```

(continues on next page)

(continued from previous page)

```
Positional arguments:
  --tolerance <TOLERANCE>        Distance tolerance for simplification. [required]

Options:
  --active-layer <ACTIVE-LAYER>  Set active layer (if not specified, all)
  --preserve-boundary            Whether the exterior boundary should be preserved.
```

Details for options can be found in *gdal vector simplify-coverage*.

sort

```
* sort [OPTIONS]
----------------

Spatially order the features in a layer

Options:
  --geometry-field <GEOMETRY-FIELD>  Name of geometry field to use in sort
  --method <METHOD>                  Geometry sorting algorithm. METHOD=hilbert|strtree␣
↪(default: hilbert)
  --use-tempfile                     Write features to a temporary file to avoid reading␣
↪the entire input dataset into memory
```

Details for options can be found in *gdal vector sort*.

sql

```
* sql [OPTIONS] <statement>|@<filename>
---------------------------------------

Apply SQL statement(s) to a dataset.

Positional arguments:
  --sql <statement>|@<filename>      SQL statement(s) [may be repeated] [required]

Options:
  -l, --output-layer <OUTPUT-LAYER>  Output layer name(s) [may be repeated]
  --dialect <DIALECT>                SQL dialect (e.g. OGRSQL, SQLITE)
```

Details for options can be found in *gdal vector sql*.

swap-xy

```
* swap-xy [OPTIONS]
-------------------

Swap X and Y coordinates of geometries of a vector dataset.

Options:
  --active-layer <ACTIVE-LAYER>        Set active layer (if not specified, all)
  --active-geometry <ACTIVE-GEOMETRY>  Geometry field name to which to restrict the␣
↪processing (if not specified, all)
```

Details for options can be found in *gdal vector swap-xy*.

tee

```
* tee [OPTIONS] [<PIPELINE>...]
------------------------------

Pipes the input into the output stream and side nested pipelines.

Positional arguments:
  --tee-pipeline <PIPELINE>  Nested pipeline [1.. values]
```

Details for options can be found in *Output nested pipeline*.

update

```
* update [OPTIONS] <OUTPUT>
---------------------------

Update an existing vector dataset with an input vector dataset.

Positional arguments:
  -o, --output <OUTPUT>                           Output vector dataset [required]

Options:
  --input-layer <INPUT-LAYER>                     Input layer name
  --output-layer <OUTPUT-LAYER>                   Output layer name
  --mode <MODE>                                   Set update mode. MODE=merge|update-
↪only|append-only (default: merge)
  --key <KEY>                                     Field(s) used as a key to identify␣
↪features [may be repeated]

Advanced Options:
  --output-oo, --output-open-option <KEY>=<VALUE>  Output open options [may be repeated]
```

Details for options can be found in *gdal vector update*.

write

```
* write [OPTIONS] <OUTPUT>
--------------------------

Write a vector dataset.

Positional arguments:
  -o, --output <OUTPUT>                                Output vector dataset [required]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)
  --co, --creation-option <KEY>=<VALUE>                Creation option [may be repeated]
  --lco, --layer-creation-option <KEY>=<VALUE>         Layer creation option [may be␣
↪repeated]
```

(continues on next page)

(continued from previous page)

```
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed
  --update                                               Whether to open existing dataset␣
↪in update mode
  --overwrite-layer                                      Whether overwriting existing␣
↪output layer is allowed
  --append                                               Whether appending to existing␣
↪layer is allowed
                                                         Mutually exclusive with --upsert
  -l, --output-layer <OUTPUT-LAYER>                      Output layer name
  --skip-errors                                          Skip errors when writing features
  --no-create-empty-layers                               Avoid creating layers to which no␣
↪features will be written

Advanced Options:
  --output-oo, --output-open-option <KEY>=<VALUE>        Output open options [may be␣
↪repeated]
  --upsert                                               Upsert features (implies 'append')
                                                         Mutually exclusive with --append
```

Details for options can be found in *gdal vector pipeline write*.

GDALG output (on-the-fly / streamed dataset)

A pipeline can be serialized as a JSON file using the `GDALG` output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

The `command_line` member of the JSON file should nominally be the whole command line without the final `write` step, and is what is generated by `gdal vector pipeline ! .... ! write out.gdalg.json`.

```
{
    "type": "gdal_streamed_alg",
    "command_line": "gdal vector pipeline ! read in.gpkg ! reproject --output-
↪crs=EPSG:32632"
}
```

The final `write` step can be added but if so it must explicitly specify the `stream` output format and a non-significant output dataset name.

```
{
    "type": "gdal_streamed_alg",
    "command_line": "gdal vector pipeline ! read in.gpkg ! reproject --output-
↪crs=EPSG:32632 ! write --output-format=streamed streamed_dataset"
}
```

Substitutions

Added in version 3.12.

It is possible to use **`gdal pipeline`** to use a pipeline already serialized in a .gdal.json file, and customize its existing steps, typically changing an input filename, specifying an output filename, or adding/modifying arguments of steps.

See *Substitutions*.

Nested pipeline

Added in version 3.12.

It is possible to create "nested pipelines", i.e. pipelines inside pipelines.

A nested pipeline is delimited by square brackets (`[` and `]`) surrounded by a space character.

There are 2 kinds of nested pipelines:

- input nested pipelines: where the result dataset of the nested pipeline is used as the input dataset for an argument of the main pipeline.
- output nested pipelines: where the output of a step of the main pipeline is used as the input of the nested pipeline in a following step. Output nested pipelines can only be used with the `tee` step.

See *Nested pipeline*.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Reproject a GeoPackage file to CRS EPSG:32632 ("WGS 84 / UTM zone 32N")

```
$ gdal vector pipeline ! read in.gpkg ! reproject --output-crs=EPSG:32632 ! write out.
↪gpkg --overwrite
```

Example 2: Serialize the command of a reprojection of a GeoPackage file in a GDALG file, and later read it

```
$ gdal vector pipeline ! read in.gpkg ! reproject --output-crs=EPSG:32632 ! write in_
↪epsg_32632.gdalg.json --overwrite
$ gdal vector info in_epsg_32632.gdalg.json
```

Example 3: Union 2 source shapefiles (with similar structure), reproject them to EPSG:32632, keep only cities larger than 1 million inhabitants and write to a GeoPackage

```
$ gdal vector pipeline ! concat --single --output-crs=EPSG:32632 france.shp belgium.shp !
↪ filter --where "pop > 1e6" ! write out.gpkg --overwrite
```

Example 4: Filter and reproject a GeoPackage layer to a GeoJSON file

When using *select* to restrict the output fields, ensure that it appears **after** the *filter* step in the pipeline when the filter expression references fields. If a field required by the filter is removed by `select`, it will not be available and the command will fail with: `ERROR 1: "worldcity" not recognised as an available field.`

If you want the output to contain spatial data, you must also include the geometry field in the selected fields. In this example the source dataset has no named geometry column. In such cases the special field `_ogr_geometry_` can be used to reference the geometry. If no geometry field is included, reprojection will fail with: `ERROR 1: reproject: Layer 'ne_110m_populated_places_simple' has no spatial reference system`.

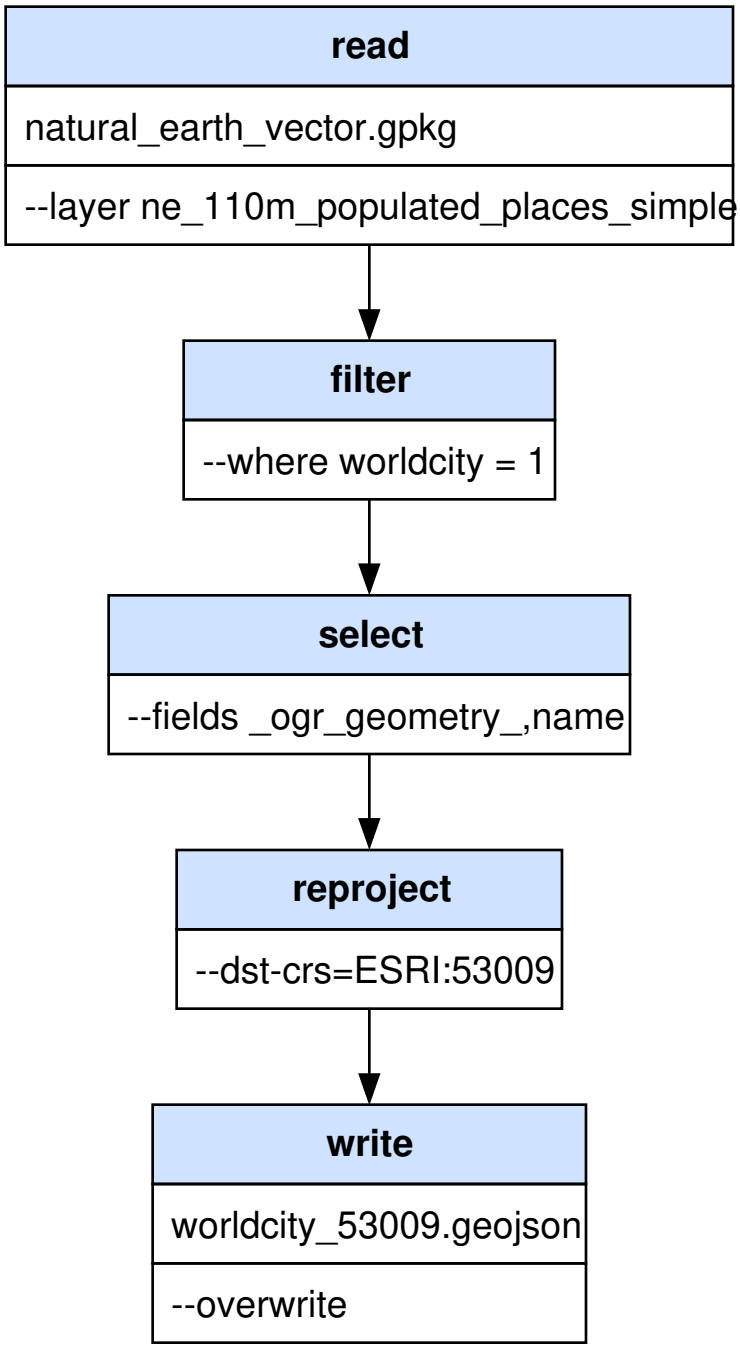
Finally, ensure that there are **no spaces** between the field names in the list. For example: `"_ogr_geometry_, name"` will produce the error: `ERROR 1: Field ' name' does not exist in layer 'ne_110m_populated_places_simple'`.

Bash

PowerShell

```
gdal vector pipeline \
    ! read natural_earth_vector.gpkg --layer ne_110m_populated_places_simple \
    ! filter --where "worldcity = 1" \
    ! select --fields "_ogr_geometry_,name" \
    ! reproject --output-crs=ESRI:53009 \
    ! write worldcity_53009.geojson --overwrite
```

```
gdal vector pipeline `
    ! read natural_earth_vector.gpkg --layer ne_110m_populated_places_simple `
    ! filter --where "worldcity = 1" `
    ! select --fields "_ogr_geometry_,name" `
    ! reproject --output-crs=ESRI:53009 `
    ! write worldcity_53009.geojson --overwrite
```



Example 5: Clip a vector layer using a nested pipeline as the clipping source

Bash

PowerShell

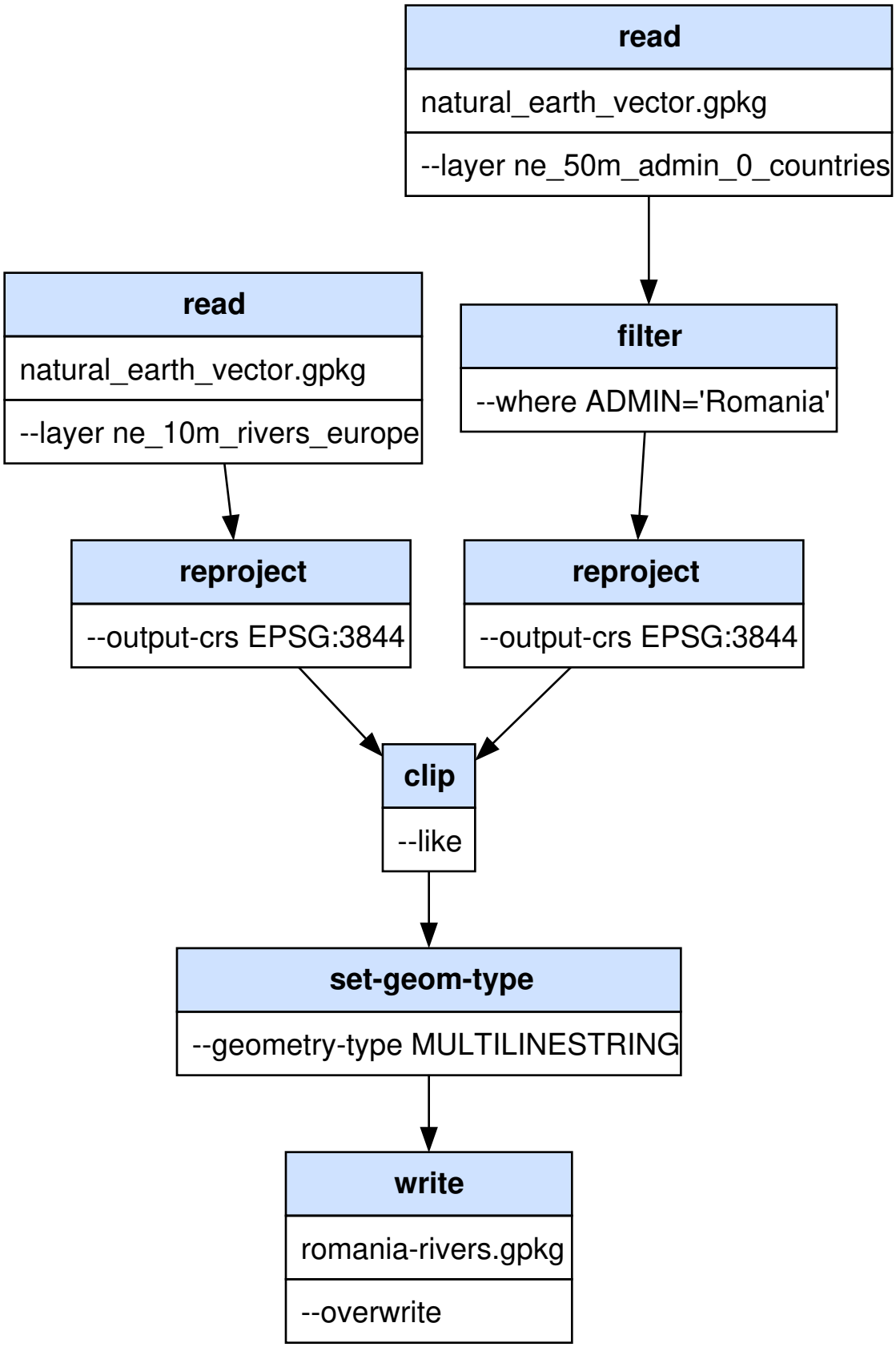
```
gdal vector pipeline \
 ! read natural_earth_vector.gpkg --layer "ne_10m_rivers_europe" \
 ! reproject --output-crs="EPSG:3844" \
 ! clip --like [ read natural_earth_vector.gpkg --layer "ne_50m_admin_0_countries" !␣
```

(continues on next page)

(continued from previous page)

```
↪filter --where "ADMIN='Romania'" ! reproject --output-crs="EPSG:3844" ] \
 ! set-geom-type --geometry-type="MULTILINESTRING" \
 ! write romania-rivers.gpkg --overwrite
```

```
gdal vector pipeline `
 ! read natural_earth_vector.gpkg --layer "ne_10m_rivers_europe" `
 ! reproject --output-crs="EPSG:3844" `
 ! clip --like [ read natural_earth_vector.gpkg --layer "ne_50m_admin_0_countries" !␣
↪filter --where "ADMIN='Romania'" ! reproject --output-crs="EPSG:3844" ] `
 ! set-geom-type --geometry-type="MULTILINESTRING" `
 ! write romania-rivers.gpkg --overwrite
```


read
natural_earth_vector.gpkg
--layer ne_50m_admin_0_countries
read
natural_earth_vector.gpkg
--layer ne_10m_rivers_europe
filter
--where ADMIN='Romania'
reproject
--output-crs EPSG:3844
reproject
--output-crs EPSG:3844
clip
--like
set-geom-type
--geometry-type MULTILINESTRING
write
romania-rivers.gpkg
--overwrite

5.1.4.27 gdal vector rasterize

Added in version 3.11.

Synopsis

```
Usage: gdal vector rasterize [OPTIONS] <INPUT> <OUTPUT>

Burns vector geometries into a raster.

Positional arguments:
  -i, --input <INPUT>                                       Input vector datasets␣
↪[required]
  -o, --output <OUTPUT>                                     Output raster dataset␣
↪[required]

Common Options:
  -h, --help                                                Display help message and exit
  --json-usage                                              Display usage as JSON␣
↪document and exit
  --config <KEY>=<VALUE>                                    Configuration option [may be␣
↪repeated]
  -q, --quiet                                               Quiet mode (no progress bar␣
↪or warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>       Output format
  --co, --creation-option <KEY>=<VALUE>                     Creation option [may be␣
↪repeated]
  --overwrite                                               Whether overwriting existing␣
↪output dataset is allowed
  -b, --band <BAND>                                         The band(s) to burn values␣
↪into (1-based index) [may be repeated]
  --invert                                                  Invert the rasterization
  --all-touched                                             Enables the ALL_TOUCHED␣
↪rasterization option
  --burn <BURN>                                             Burn value [may be repeated]
  -a, --attribute-name <ATTRIBUTE-NAME>                     Attribute name
  --3d                                                      Indicates that a burn value␣
↪should be extracted from the Z values of the feature
  -l, --input-layer <INPUT-LAYER>                           Input layer name
                                                            Mutually exclusive with --sql
  --where <WHERE>                                           SQL where clause
  --sql <SQL>                                               SQL select statement
                                                            Mutually exclusive with --
↪input-layer
  --dialect <DIALECT>                                       SQL dialect
  --nodata <NODATA>                                         Assign a specified nodata␣
↪value to output bands
  --init <INIT>                                             Pre-initialize output bands␣
↪with specified value [may be repeated]
  --crs <CRS>                                               Override the projection for␣
↪the output file
```

(continues on next page)

(continued from previous page)

```
  --transformer-option <NAME>=<VALUE>                      Set a transformer option␣
↪suitable to pass to GDALCreateGenImgProjTransformer2 [may be repeated]
  --extent <xmin>,<ymin>,<xmax>,<ymax>                     Set the target georeferenced␣
↪extent [4 values]
  --resolution <xres>,<yres>                               Set the target resolution [2␣
↪values]
                                                           Mutually exclusive with --size
  --tap, --target-aligned-pixels                           (target aligned pixels) Align␣
↪the coordinates of the extent of the output file to the values of the resolution
                                                           Depends on --resolution
  --size <xsize>,<ysize>                                   Set the target size in pixels␣
↪and lines [2 values]
                                                           Mutually exclusive with --
↪resolution
  --ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>  Output data type. OUTPUT-DATA-
↪TYPE=UInt8|Int8|UInt16|Int16|UInt32|Int32|UInt64|Int64|CInt16|CInt32|Float16|Float32|Float64|CFloat32
  --optimization <OPTIMIZATION>                            Force the algorithm used␣
↪(results are identical). OPTIMIZATION=AUTO|RASTER|VECTOR (default: AUTO)
  --add                                                    Add to existing raster
  --update                                                 Whether to open existing␣
↪dataset in update mode

Advanced Options:
  --oo, --open-option <KEY>=<VALUE>                        Open options [may be repeated]
  --if, --input-format <INPUT-FORMAT>                      Input formats [may be␣
↪repeated]
```

Description

gdal vector rasterize burns vector geometries into a raster.

By default (unless `--update` or `--add`) is specified, a new raster is created with a specified resolution (`--resolution`) or size (`--size`). Pixel values will either be set to a constant (the value 1, or some other value specified with `--burn`), the Z value of feature geometries (with `--3d`), or the value of a field specified with `--attribute-name`.

The left figure illustrates the default rasterization behavior, where only pixels on the line render path are included.

The right figure shows the effect of `--all-touched`, where every pixel that touches the input polygons is included.

Since GDAL 3.12, this algorithm can be part of a *gdal pipeline*.

The following options are available:

Added in version 3.12.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a raster dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Note

However this algorithm is not natively streaming compatible. Consequently a temporary dataset will be generated, which may cause significant processing time at opening.

Program-Specific Options

--3d

Indicates that a burn value should be extracted from the "Z" values of the feature. Works with points and lines (linear interpolation along each segment). For polygons, works properly only if they are flat (same Z value for all vertices).

--add

Instead of burning a new value, this adds the new value to the existing raster, implies `--update`. Suitable for heatmaps for instance.

--all-touched

Enables the ALL_TOUCHED rasterization option.

> **Note**
>
> When this option is enabled, the order of the input features (lines or polygons) can affect the results. When two features touch each other, the last one (i.e. topmost) will determine the burned pixel value at the edge. You may wish to use the *--sql* option to reorder the features (ORDER BY) to achieve a more predictable result.

-a, --attribute-name <ATTRIBUTE-NAME>

Name of the attribute / field from which a burn value should be retrieved for each feature. GDAL will attempt to parse the field value as a number. If this is not successful, a value of zero will be written.

--band, -b <BAND>

The band(s) to burn values into. May be repeated.

--burn <BURN>

Burn value. May be repeated.

--crs <CRS>

Override the projection for the output file. If not specified, the projection of the input vector file will be used if available. When using this option, no reprojection of features from the CRS of the input vector to the specified CRS of the output raster, so use only this option to correct an invalid source CRS. The <CRS> may be any of the usual GDAL/OGR forms, complete WKT, PROJ.4, EPSG:n or a file containing the WKT.

--extent <xmin>,<ymin>,<xmax>,<ymax>

Set georeferenced extents. The values must be expressed in georeferenced units. If not specified, the extent of the output file will be the extent of the vector layers.

--init <INIT>

Pre-initialize output bands with specified value. May be repeated.

-l, --input-layer <INPUT-LAYER>

Indicates the layer(s) from the datasource that will be used for input features. May be specified multiple times, but at least one layer name or a -sql option must be specified (not both).

--invert

Invert rasterization. Burn the fixed burn value, or the burn value associated with the first feature into all parts of the image not inside the provided polygon.

> **Note**
>
> When the vector features contain a polygon nested within another polygon (like an island in a lake), GDAL must be built against GEOS to get correct results.

--nodata <NODATA>

Assign a specified nodata value to output bands.

--optimization <OPTIMIZATION>

Force the algorithm used (results are identical). The raster mode is used in most cases and optimise read/write operations. The vector mode is useful with a decent amount of input features and optimise the CPU use. That mode have to be used with tiled images to be efficient. The auto mode (the default) will chose the algorithm based on input and output properties.

--ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>

Force the output bands to be of the indicated data type. Defaults to `Float64`, unless the attribute field to burn is of type `Int64`, in which case `Int64` is used for the output raster data type if the output driver supports it.

--resolution <xres>,<yres>

Set target resolution. The values must be expressed in georeferenced units. Both must be positive values. Note that *--resolution* cannot be used with *--size*.

--size <xsize>,<ysize>

Set output file size in pixels and lines. Note that *--size* cannot be used with *--resolution*. If one of the two values is set to 0, it will be computed from the other value in order to preserve the aspect ratio of the extent.

--sql <SQL>|@<filename>

An SQL statement to be evaluated against the datasource to produce a virtual layer of features to be burned in. The @filename syntax can be used to indicate that the content is in the pointed filename.

--dialect <DIALECT>

SQL dialect.

By default the native SQL of an RDBMS is used when using `gdal vector sql`. If using `sql` as a step of `gdal vector pipeline`, this is only true if the step preceding `sql` is `read`, otherwise the *OGRSQL* dialect is used.

If a datasource does not support SQL natively, the default is to use the `OGRSQL` dialect, which can also be specified with any data source.

The *SQL SQLite dialect* dialect can be chosen with the `SQLITE` and `INDIRECT_SQLITE` dialect values, and this can be used with any data source. Overriding the default dialect may be beneficial because the capabilities of the SQL dialects vary.

Supported dialects can be checked with `gdal --format`. For example:

```
$ gdal --format "PostgreSQL"
[...]
Supported SQL dialects: NATIVE OGRSQL SQLITE
[...]

$ gdal --format "ESRI Shapefile"
[...]
Supported SQL dialects: OGRSQL SQLITE
[...]
```

--tap, --target-aligned-pixels

(target aligned pixels) Align the coordinates of the extent of the output file to the values of *--resolution*, such that the aligned extent includes the minimum extent. Alignment means that xmin / resx, ymin / resy, xmax / resx and ymax / resy are integer values. `--tap` requires `--resolution`.

--transformer-option <NAME>=<VALUE>

set a transformer option suitable to pass to GDALCreateGenImgProjTransformer2(). This is used when converting geometries coordinates to target raster pixel space. For example this can be used to specify RPC related transformer options.

--where <WHERE>

An optional SQL WHERE style query expression to be applied to select features to burn in from the input layer(s).

Standard Options

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output raster format to use. Allowed values may be given by `gdal --formats | grep raster | grep rw | sort`

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--update

Whether to open an existing output dataset in update mode.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Burn a shapefile into a raster

The following would burn all polygons from `mask.shp` into the RGB TIFF file `work.tif` with the color red (RGB = 255,0,0).

```
gdal vector rasterize -b 1,2,3 --burn 255,0,0 -l mask mask.shp work.tif
```

Example 2: The following would burn all "class A" buildings into the output elevation file, pulling the top elevation from the ROOF_H attribute.

Burn a shapefile into a raster using a specific where condition to select features

```
gdal vector rasterize -a ROOF_H --where "class='A'" -l footprints footprints.shp city_
↪dem.tif
```

Example 3: The following would burn all polygons from footprint.shp into a new 1000x1000 rgb TIFF as the color red.

> **Note**
>
> *-b* is not used; the order of the *--burn* options determines the bands of the output raster.

```
gdal vector rasterize --burn 255,0,0 --ot Byte --size 1000,1000 -l footprints footprints.
↪shp mask.tif
```

Example 4: Burn a shapefile into a raster using a specific where condition to select features, and restrict the extent to the one of selected features

```
gdal pipeline read /vsizip/vsicurl/https://www2.census.gov/geo/tiger/TIGER2025/STATE/tl_
↪2025_us_state.zip ! \
                filter --where  "stusps = 'CA'" --update-extent ! \
                rasterize  --burn 1 --size 1500,1500 --datatype Byte ! \
                write out.png --overwrite
```

5.1.4.28 `gdal vector pipeline read`

Added in version 3.11.

Description

The `read` operation is for use in a *gdal pipeline* only, and reads a single input vector dataset. This is the first step of a pipeline.

To test simple spatial operations, the `read` step can read a single geometry in (extended) WKT format rather than a dataset.

Synopsis

```
* read [OPTIONS] <INPUT>
------------------------

Read a vector dataset.

Positional arguments:
  -i, --input <INPUT>                       Input vector datasets [required]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>  Input layer name(s) [may be repeated]

Advanced Options:
  --if, --input-format <INPUT-FORMAT>       Input formats [may be repeated]
  --oo, --open-option <KEY>=<VALUE>         Open options [may be repeated]
```

Standard Options

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

Examples

Example 1: Read a GeoPackage file

```
$ gdal vector pipeline read input.gpkg ! ... [other commands here] ...
```

Example 2: Read and buffer a single point

```
$ gdal vector pipeline read "SRID=32145;POINT (442922 217537)" ! buffer 20 ! ... [other␣
↪commands here] ...
```

5.1.4.29 gdal vector rename-layer

Added in version 3.13.

Synopsis

```
Usage: gdal vector rename-layer [OPTIONS] <INPUT> <OUTPUT>

Rename layer(s) of a vector dataset.
```

(continues on next page)

(continued from previous page)

```
Positional arguments:
  -i, --input <INPUT>                                   Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                 Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                            Display help message and exit
  --json-usage                                          Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                Configuration option [may be␣
↪repeated]
  -q, --quiet                                           Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                 Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>          Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                           Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                              Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                     Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                              Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                        Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                         Output layer name
  --skip-errors                                         Skip errors when writing features␣
↪[not available in pipelines]
  -l, --input-layer <INPUT-LAYER>                       Input layer name
  --ascii                                               Force names to ASCII character
  --lower-case                                          Force names to lower case (only␣
↪on ASCII characters)
  --filename-compatible                                 Force names to be usable as␣
↪filenames
  --reserved-characters <RESERVED-CHARACTERS>           Reserved character(s) to be␣
↪removed
  --replacement-character <REPLACEMENT-CHARACTER>       Replacement character when ASCII␣
↪conversion not possible
  --max-length <MAX-LENGTH>                             Maximum length of layer names

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                   Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                     Open options [may be repeated]␣
↪[not available in pipelines]
```

(continues on next page)

(continued from previous page)

```
  --output-oo, --output-open-option <KEY>=<VALUE>      Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                             Upsert features (implies 'append
↪') [not available in pipelines]
                                                       Mutually exclusive with --append
```

Description

gdal vector rename-layer can be used to rename layer(s).

It has two modes:

- one where the user renames a single layer to a given name with *--output-layer*, combined with *--input-layer* if there are more than one layer.
- another one where the user may force various operations to alter the layer names:
  - Force characters to ASCII: *--ascii*
  - Force characters to lower case: *--lower-case*
  - Force compatibility with file names: *--filename-compatible*
  - Remove or replace characters from a list of reserved characters: *--reserved-characters*
  - Truncate to a maximum number of characters: *--max-length*

  For *--filename-compatible* and *--reserved-characters*, incompatible characters are removed, unless *--replacement-character* is specified.

  In this mode, uniqueness of layer names is also attempted by appending `_{N}` where NN is a sequence number. Uniqueness may not be achieved if *--max-length* is set to a too low value.

This subcommand is also available as a potential step of *gdal vector pipeline*

Program Specific Options

--input-layer <INPUT-LAYER>

Name of input layer. To be used together with *--output-layer*

--output-layer <OUTPUT-LAYER>

Name of output layer. If there are several input layers, *--output-layer* must also be specified.

--ascii

Force characters to ASCII. Currently only accented characters characters in Latin 1 Supplement and Latin Extended A character set have a sensible ASCII replacement. Other non-ASCII characters will be removed or replaced by the replacement character specified with *--replacement-character*

--lower-case

Force characters to lower-case. Currently only ASCII upper case characters are replaced. This option is applied after *--ascii*, if the later is specified.

--filename-compatible

Force layer names to be compatible with file names for Linux, Windows or MacOS, independently from the operating system on which this command is run.

--reserved-characters <RESERVED-CHARACTERS>

Reserved character(s) to remove or replace with the replacement character specified with *--replacement-character*

--replacement-character <REPLACEMENT-CHARACTER>

Replacement character when ASCII conversion is not possible or to substitute to reserved characters.

--max-length <MAX-LENGTH>

Truncate to a maximum number of characters

Standard Options

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-layer <NAME>

Specifies the name of the layer to which features will be written. If not specified, output layer names will be the same as input layer names.

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Rename a given layer to a user selected name.

```
$ gdal vector rename-layer input.gpkg output.gpkg --input-layer "name with space" --
↪output-layer "without_space"
```

Example 2: Force layer names to ASCII, lower case, without space character and to be filename compatible.

```
$ gdal vector rename-layer input.gpkg output_shp_directory --output-format "ESRI␣
↪Shapefile" --ascii --lower-case --filename-compatible --reserved-characters " "
```

5.1.4.30 gdal vector reproject

Added in version 3.11.

Synopsis

```
Usage: gdal vector reproject [OPTIONS] <INPUT> <OUTPUT>

Reproject a vector dataset.

Positional arguments:
  -i, --input <INPUT>                                    Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>               Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>           Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                               Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                      Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                               Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                         Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                          Output layer name [not available␣
↪in pipelines]
  --skip-errors                                          Skip errors when writing features␣
↪[not available in pipelines]
  --active-layer <ACTIVE-LAYER>                          Set active layer (if not␣
↪specified, all)
```

(continues on next page)

(continued from previous page)

```
  -s, --input-crs <INPUT-CRS>                          Input CRS
  -d, --output-crs <OUTPUT-CRS>                        Output CRS [required]

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                  Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                    Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>      Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                             Upsert features (implies 'append
↪') [not available in pipelines]
                                                       Mutually exclusive with --append
```

Description

gdal vector reproject can be used to reproject a vector dataset. The program can reproject to any supported projection.

This command can also be used as a step of *gdal vector pipeline*.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

--input-crs, -s, <INPUT-CRS>

Override input spatial reference. If not specified the SRS found in the input dataset will be used.

The coordinate reference systems that can be passed are anything supported by the OGRSpatialReference.SetFromUserInput() call, which includes EPSG Projected, Geographic or Compound CRS (i.e. EPSG:4296), a well known text (WKT) CRS definition, PROJ.4 declarations, or the name of a .prj file containing a WKT CRS definition.

If the SRS has an explicit vertical datum that points to a PROJ.4 geoidgrids, and the input dataset is a single band dataset, a vertical correction will be applied to the values of the dataset.

--output-crs, -d <OUTPUT-CRS>

Set output spatial reference.

The coordinate reference systems that can be passed are anything supported by the OGRSpatialReference.SetFromUserInput() call, which includes EPSG Projected, Geographic or Compound CRS (i.e. EPSG:4296), a well known text (WKT) CRS definition, PROJ.4 declarations, or the name of a .prj file containing a WKT CRS definition.

If the SRS has an explicit vertical datum that points to a PROJ.4 geoidgrids, and the input dataset is a single band dataset, a vertical correction will be applied to the values of the dataset.

Standard Options

--active-layer <ACTIVE-LAYER>

Set the active layer. When it is specified, only the layer specified by its name will be subject to the processing. Other layers will be not modified. If this option is not specified, all layers will be subject to the processing.

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--input-layer <INPUT-LAYER>

Specifies the name of one or more layers to process. By default, all layers will be processed. To read and write all layers but only process a subset, use `--active-layer`.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-layer <NAME>

Specifies the name of the layer to which features will be written. If not specified, output layer names will be the same as input layer names.

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

-q, --quiet

Suppress progress bar and some warning messages.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Reproject a GeoPackage file to CRS EPSG:32632 ("WGS 84 / UTM zone 32N")

```
$ gdal vector reproject --output-crs=EPSG:32632 in.gpkg out.gpkg --overwrite
```

Example 2: Reproject data using various CRS formats

The following examples demonstrate different ways to specify a coordinate reference system. Each command reprojects the data to the Web Mercator (EPSG:3857) projection and produces identical output.

```
# OGC CRS URI
$ gdal vector reproject \
    --output-crs="http://www.opengis.net/def/crs/EPSG/0/3857" \
    natural_earth_vector.gpkg --layer=ne_10m_populated_places \
    places.json --overwrite

# OGC CRS URN
$ gdal vector reproject \
    --output-crs="urn:ogc:def:crs:EPSG::3857" \
    natural_earth_vector.gpkg --layer=ne_10m_populated_places \
    places.json --overwrite

# PROJ string (legacy format)
$ PROJ4="+proj=merc +a=6378137 +b=6378137 +lat_ts=0 +lon_0=0 +x_0=0 +y_0=0 +k=1 +units=m␣
↪+nadgrids=@null +wktext +no_defs +type=crs"
$ gdal vector reproject \
    --output-crs="$PROJ4" \
    natural_earth_vector.gpkg --layer=ne_10m_populated_places \
    places.json --overwrite
```

Example 3: Reproject a layer in a GeoPackage to Web Mercator GeoJSON

The timezone layer cannot be fully reprojected to Web Mercator as the coordinates at the poles fall outside the extent of the Web Mercator bounding-box causing some features to be missing in the output. The following errors are returned:

```
ERROR 1: Full reprojection failed, but partial is possible if you define OGR_ENABLE_
↪PARTIAL_REPROJECTION configuration option to TRUE
ERROR 1: PROJ: webmerc: Invalid latitude
ERROR 1: Reprojection failed, err = 2049, further errors will be suppressed on the␣
↪transform object.
```

You can follow the suggestion above and enable `OGR_ENABLE_PARTIAL_REPROJECTION`. Errors from PROJ relating to invalid coordinates (`ERROR 1: PROJ: webmerc: Invalid latitude`) will still be reported, but all features will be written to the output.

```
$ gdal vector reproject \
    --output-crs=EPSG:3857 \
    --config OGR_ENABLE_PARTIAL_REPROJECTION=TRUE \
    natural_earth_vector.gpkg --layer=ne_10m_time_zones \
    ne_10m_time_zones.json \
    --overwrite
```

5.1.4.31 `gdal vector select`

Added in version 3.11.

Synopsis

```
Usage: gdal vector select [OPTIONS] <INPUT> <OUTPUT> <FIELDS>

Select a subset of fields from a vector dataset.

Positional arguments:
  -i, --input <INPUT>                                    Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]
  --fields <FIELDS>                                      Fields to select (or exclude if --
↪exclude) [may be repeated] [required]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>               Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>           Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                               Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                      Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                               Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                         Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                          Output layer name
  --skip-errors                                          Skip errors when writing features␣
↪[not available in pipelines]
  --active-layer <ACTIVE-LAYER>                          Set active layer (if not␣
↪specified, all)
  --exclude                                              Exclude specified fields
                                                         Mutually exclusive with --ignore-
↪missing-fields
  --ignore-missing-fields                                Ignore missing fields
                                                         Mutually exclusive with --exclude

Advanced Options:
```

(continues on next page)

(continued from previous page)

```
  --if, --input-format <INPUT-FORMAT>                    Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                      Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>        Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                               Upsert features (implies 'append
↪') [not available in pipelines]
                                                         Mutually exclusive with --append
```

Description

gdal vector select can be used to select a subset of fields.

`select` can also be used as a step of *gdal vector pipeline*.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

--exclude

Modifies the behavior of the algorithm such that all fields are selected, except the ones mentioned by *--fields*.

--fields <FIELDS>

Comma-separated list of fields from input layer to copy to the new layer (or to exclude if *--exclude* is specified)

Field names with spaces, commas or double-quote should be surrounded with a starting and ending double-quote character, and double-quote characters in a field name should be escaped with backslash.

Depending on the shell used, this might require further quoting. For example, to select `regular_field`, `a_field_with space, and comma` and `a field with " double quote` with a Unix shell:

```
--fields "regular_field,\"a_field_with space, and comma\",\"a field with \\\"␣
↪double quote\""
```

A field is only selected once, even if mentioned several times in the list.

Geometry fields can also be specified in the list. If the source layer has no explicit name for the geometry field, `_ogr_geometry_` must be used to select the unique geometry field.

Specifying a non-existing source field name results in an error.

--ignore-missing-fields

By default, if a field specified by *--fields* does not exist in the input layer(s), an error is emitted and the processing is stopped. When specifying *--ignore-missing-fields*, only a warning is emitted and the non existing fields are just ignored.

Standard Options

--active-layer <ACTIVE-LAYER>

Set the active layer. When it is specified, only the layer specified by its name will be subject to the processing. Other layers will be not modified. If this option is not specified, all layers will be subject to the processing.

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--input-layer <INPUT-LAYER>

Specifies the name of one or more layers to process. By default, all layers will be processed. To read and write all layers but only process a subset, use `--active-layer`.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-layer <NAME>

Specifies the name of the layer to which features will be written. If not specified, output layer names will be the same as input layer names.

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Select the EAS_ID field and the geometry field from a Shapefile

```
$ gdal vector select in.shp out.gpkg "EAS_ID,_ogr_geometry_" --overwrite
```

Example 2: Remove sensitive fields from a layer

```
$ gdal vector select in.shp out.gpkg --exclude "name,surname,address" --overwrite
```

5.1.4.32 gdal vector segmentize

Added in version 3.12.

Synopsis

```
Usage: gdal vector segmentize [OPTIONS] <INPUT> <OUTPUT> <MAX-LENGTH>

Segmentize geometries of a vector dataset.

Positional arguments:
  -i, --input <INPUT>                                    Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]
  --max-length <MAX-LENGTH>                              Maximum length of a segment␣
↪[required]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>               Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>           Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                               Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                      Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                               Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                         Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                          Output layer name [not available␣
↪in pipelines]
  --skip-errors                                          Skip errors when writing features␣
↪[not available in pipelines]
```

(continues on next page)

(continued from previous page)

```
  --active-layer <ACTIVE-LAYER>                          Set active layer (if not␣
↪specified, all)
  --active-geometry <ACTIVE-GEOMETRY>                    Geometry field name to which to␣
↪restrict the processing (if not specified, all)

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                    Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                      Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>        Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                               Upsert features (implies 'append
↪') [not available in pipelines]
                                                         Mutually exclusive with --append
```

Description

gdal vector segmentize ensures segments of LineStrings or rings of Polygons are not longer than the specified max length, by inserting intermediate points.

It runs the *OGRGeometry::segmentize()* operation.

It can also be used as a step of *gdal vector pipeline*.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the GDALG output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

--max-length <MAX-LENGTH>

The specified value of this option is the maximum distance between two consecutive points of the output geometry before intermediate points are added. The unit of the distance is georeferenced units of the source layer.

Standard Options

--active-geometry <ACTIVE-GEOMETRY>

Set the active geometry field from its name. When it is specified, only the specified geometry field will be subject to the processing. Other geometry fields will be not modified. If this option is not specified, all geometry fields will be subject to the processing. This option can be combined together with *--active-layer*.

--active-layer <ACTIVE-LAYER>

Set the active layer. When it is specified, only the layer specified by its name will be subject to the processing. Other layers will be not modified. If this option is not specified, all layers will be subject to the processing.

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--input-layer <INPUT-LAYER>

Specifies the name of one or more layers to process. By default, all layers will be processed. To read and write all layers but only process a subset, use `--active-layer`.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-layer <NAME>

Specifies the name of the layer to which features will be written. If not specified, output layer names will be the same as input layer names.

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of `--append` where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Make sure that segments of geometries are no longer than one km (assuming the CRS is in meters)

```
$ gdal vector segmentize --max-length=1000 in.gpkg out.gpkg --overwrite
```

5.1.4.33 gdal vector set-field-type

Added in version 3.12.

Synopsis

```
Usage: gdal vector set-field-type [OPTIONS] <INPUT> <OUTPUT>

Modify the type of a field of a vector dataset.
```

(continues on next page)

(continued from previous page)

```
Positional arguments:
  -i, --input <INPUT>                                   Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                 Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                            Display help message and exit
  --json-usage                                          Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                Configuration option [may be␣
↪repeated]
  -q, --quiet                                           Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>              Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                 Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>          Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                           Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                              Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                     Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                              Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                        Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                         Output layer name [not available␣
↪in pipelines]
  --skip-errors                                         Skip errors when writing features␣
↪[not available in pipelines]
  --active-layer <ACTIVE-LAYER>                         Set active layer (if not␣
↪specified, all)
  --field-name <FIELD-NAME>                             Field name [required]
                                                        Mutually exclusive with --input-
↪field-type
  --input-field-type <INPUT-FIELD-TYPE>                 Source field type or subtype␣
↪[required]
                                                        Mutually exclusive with --field-
↪name
  --output-field-type, --field-type <FIELD-TYPE>        Target field type or subtype␣
↪[required]

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                   Input formats [may be repeated]␣
↪[not available in pipelines]
```

(continues on next page)

(continued from previous page)

```
  --oo, --open-option <KEY>=<VALUE>                    Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>      Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                             Upsert features (implies 'append
↪') [not available in pipelines]
                                                       Mutually exclusive with --append
```

Description

gdal vector set-field-type can be used to modify the field type of a vector dataset:

`set-field-type` can also be used as a step of *gdal vector pipeline*.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

--field-name <FIELD-NAME>

The name of the field to modify.

Mutually exclusive with *--input-field-type*. One of them is required.

--field-type, --output-field-type <FIELD-TYPE>

The new field type. Valid values are: `Integer`, `IntegerList`, `Real`, `RealList`, `String`, `StringList`, `Binary`, ` `Date`, `Time`, `DateTime`, `Integer64`, `Integer64List`. A field subtype can be specified instead of a field type. Valid values are: `Boolean`, `Int16`, `Float32`, `JSON`, `UUID`. The field type will be derived from the subtype.

--input-field-type <FIELD-TYPE>

The field type of the fields to modify. Valid values are: `Integer`, `IntegerList`, `Real`, `RealList`, `String`, `StringList`, `Binary`, ` `Date`, `Time`, `DateTime`, `Integer64`, `Integer64List`. A field subtype can be specified instead of a field type. Valid values are: `Boolean`, `Int16`, `Float32`, `JSON`, `UUID`.

Mutually exclusive with *--field-name*. One of them is required.

Standard Options

--active-layer <ACTIVE-LAYER>

Set the active layer. When it is specified, only the layer specified by its name will be subject to the processing. Other layers will be not modified. If this option is not specified, all layers will be subject to the processing.

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--input-layer <INPUT-LAYER>

Specifies the name of one or more layers to process. By default, all layers will be processed. To read and write all layers but only process a subset, use `--active-layer`.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-layer <NAME>

Specifies the name of the layer to which features will be written. If not specified, output layer names will be the same as input layer names.

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Change the type of a field given by its name to Integer

```
$ gdal vector set-field-type input.gpkg output.gpkg --field-name myfield --field-type␣
↪Integer
```

Example 2: Change the type of all fields of type Date to DateTime

```
$ gdal vector set-field-type input.gpkg output.gpkg --input-field-type Date --output-
↪field-type DateTime
```

5.1.4.34 gdal vector set-geom-type

Added in version 3.12.

Synopsis

```
Usage: gdal vector set-geom-type [OPTIONS] <INPUT> <OUTPUT>

Modify the geometry type of a vector dataset.

Positional arguments:
```

(continues on next page)

(continued from previous page)

```
  -i, --input <INPUT>                                    Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>               Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>           Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                               Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                      Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                               Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                         Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                          Output layer name [not available␣
↪in pipelines]
  --skip-errors                                          Skip errors when writing features␣
↪[not available in pipelines]
  --active-layer <ACTIVE-LAYER>                          Set active layer (if not␣
↪specified, all)
  --active-geometry <ACTIVE-GEOMETRY>                    Geometry field name to which to␣
↪restrict the processing (if not specified, all)
  --layer-only                                           Only modify the layer geometry␣
↪type
                                                         Mutually exclusive with --feature-
↪only
  --feature-only                                         Only modify the geometry type of␣
↪features
                                                         Mutually exclusive with --layer-
↪only
  --geometry-type <GEOMETRY-TYPE>                        Geometry type
  --multi                                                Force geometries to MULTI␣
↪geometry types
                                                         Mutually exclusive with --single
```

(continues on next page)

(continued from previous page)

```
  --single                                                  Force geometries to non-MULTI␣
↪geometry types
                                                            Mutually exclusive with --multi
  --linear                                                  Convert curve geometries to␣
↪linear types
                                                            Mutually exclusive with --curve
  --curve                                                   Convert linear geometries to␣
↪curve types
                                                            Mutually exclusive with --linear
  --dim <DIM>                                               Force geometries to the specified␣
↪dimension. DIM=XY|XYZ|XYM|XYZM
  --skip                                                    Skip feature when change of␣
↪feature geometry type failed

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                       Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                         Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>           Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                                  Upsert features (implies 'append
↪') [not available in pipelines]
                                                            Mutually exclusive with --append
```

Description

gdal vector set-geom-type can be used to modify the geometry type of geometry fields at the layer and/or feature level.

It can also be used as a step of *gdal vector pipeline*.

The following groups of options can be combined together: - *--multi* / *--single* - *--linear* / *--curve* - *--dim*

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the GDALG output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

--curve

Force geometries to curve geometry types. e.g. LINESTRING ==> COMPOUNDCURVE or POLYGON ==> CURVEPOLYGON. Points are kept unmodified. This option is mutually exclusive with *--linear*.

--dim

Force geometries to the specified dimension (XY, XYZ, XYM, or XYZM). If the input geometry lacks Z or M components, they will be set to 0. This option is mutually exclusive with *--geometry-type*.

--feature-only

Only modify the geometry type of features.

--geometry-type <GEOMETRY-TYPE>

Change the geometry type to be one of GEOMETRY, POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION, CURVE, CIRCULARSTRING, COMPOUNDCURVE, SURFACE, CURVEPOLYGON, MULTICURVE, MULTISURFACE, POLYHEDRALSURFACE or TIN. Z, M or ZM suffixes can be appended to the above values to indicate the dimensionality.

This option is mutually exclusive with *--multi*, *--single*, *--linear*, *--curve*, and *--dim*.

--layer-only

Only modify the layer geometry type. A typical use case is if the layer geometry type is set to unknown and the user knows that the layer only contains geometry of a given type.

--linear

Force geometries to linear/non-curve geometry types, approximating arcs with linear segments. The linear approximation can be controlled with configuration options *OGR_ARC_STEPSIZE* and *OGR_ARC_MAX_GAP* (specified using *vector_common_options --config*). e.g. COMPOUNDCURVE ==> LINESTRING. This option is mutually exclusive with *--curve*.

--multi

Force geometries to MULTI geometry types. e.g. POINT ==> MULTIPOINT. This option is mutually exclusive with *--single*.

--single

Force geometries to non-MULTI geometry types. e.g. MULTIPOINT ==> POINT. This option is mutually exclusive with *--multi*. MultiPoint or MultiLinestring with more than one component will be kept unmodified. MultiPolygon with more than one component will be merged into a (likely invalid according to OGC Simple Features rule) single polygon, merging all their rings.

--skip

Skip feature when change of feature geometry type failed (e.g. attempting to force a Point geometry type for a LineString geometry). Otherwise the source geometry will be kept unmodified (which may make it incompatible with the output layer if it does not support mix of geometry types). This option applies both when the target geometry type is defined with *--geometry-type*, or with any of *--multi*, *--single*, *--linear*, and *--curve*.

Standard Options

--active-geometry <ACTIVE-GEOMETRY>

Set the active geometry field from its name. When it is specified, only the specified geometry field will be subject to the processing. Other geometry fields will be not modified. If this option is not specified, all geometry fields will be subject to the processing. This option can be combined together with *--active-layer*.

--active-layer <ACTIVE-LAYER>

Set the active layer. When it is specified, only the layer specified by its name will be subject to the processing. Other layers will be not modified. If this option is not specified, all layers will be subject to the processing.

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--input-layer <INPUT-LAYER>

Specifies the name of one or more layers to process. By default, all layers will be processed. To read and write all layers but only process a subset, use `--active-layer`.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-layer <NAME>

Specifies the name of the layer to which features will be written. If not specified, output layer names will be the same as input layer names.

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Convert a shapefile mixing polygons and multipolygons to a GeoPackage with multipolygons

```
$ gdal vector set-geom-type --geometry-type=MULTIPOLYGON in.shp out.gpkg --overwrite
```

Example 2: Convert a GeoPackage with curve geometries to a Shapefile (that does not support them)

```
$ gdal vector set-geom-type --linear in.gpkg out.shp --overwrite
```

5.1.4.35 gdal vector sort

Added in version 3.13.

Synopsis

```
Usage: gdal vector sort [OPTIONS] <INPUT> <OUTPUT>

Spatially order the features in a layer

Positional arguments:
  -i, --input <INPUT>                                  Input vector datasets [required]␣
```

(continues on next page)

(continued from previous page)

```
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>               Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>           Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                               Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                      Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                               Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                         Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                          Output layer name [not available␣
↪in pipelines]
  --skip-errors                                          Skip errors when writing features␣
↪[not available in pipelines]
  --geometry-field <GEOMETRY-FIELD>                      Name of geometry field to use in␣
↪sort
  --method <METHOD>                                      Geometry sorting algorithm.␣
↪METHOD=hilbert|strtree (default: hilbert)
  --use-tempfile                                         Write features to a temporary␣
↪file to avoid reading the entire input dataset into memory

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                    Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                      Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>        Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                               Upsert features (implies 'append
↪') [not available in pipelines]
                                                         Mutually exclusive with --append
```

Description

gdal vector sort sorts the features in a vector dataset according to the position of its geometries. The sorting method can be specified via *--method*, with Hilbert and Sort-Tile-Recursive (STRTree) algorithms available.

Features that have null or empty geometries will be placed at the end of the sorted dataset.

This command can also be used as a step of *gdal vector pipeline*.

> **Note**
>
> Use of the STRtree algorithm requires a GDAL build against the GEOS library.

Program-Specific Options

--geometry-field <GEOMETRY-FIELD>

The name of the geometry field by which features should be sorted.

--method <METHOD>

Specifies the sorting method. Available options are:

- hilbert : default method. Geometries are sorted according to the Hilbert code of the center point of their bounding box
- strtree : Geometries are sorted by constructing a sort-tile-recursive tree using the GEOS library and performing a depth-first iteration.

--use-tempfile

Write the input features to a temporary file from which they can be read in sorted order. Avoids the need to load the entire input layer into memory.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Create a cloud-optimized Shapefile

```
$ gdal vector sort in.gpkg out.shp --method hilbert --lco SPATIAL_INDEX=YES
```

5.1.4.36 gdal vector simplify

Added in version 3.12.

Synopsis

```
Usage: gdal vector simplify [OPTIONS] <INPUT> <OUTPUT> <TOLERANCE>

Simplify geometries of a vector dataset.

Positional arguments:
  -i, --input <INPUT>                                Input vector datasets [required]␣
```

(continues on next page)

(continued from previous page)

```
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]
  --tolerance <TOLERANCE>                                Distance tolerance for␣
↪simplification. [required]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>               Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>           Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                               Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                      Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                               Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                         Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                          Output layer name [not available␣
↪in pipelines]
  --skip-errors                                          Skip errors when writing features␣
↪[not available in pipelines]
  --active-layer <ACTIVE-LAYER>                          Set active layer (if not␣
↪specified, all)
  --active-geometry <ACTIVE-GEOMETRY>                    Geometry field name to which to␣
↪restrict the processing (if not specified, all)

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                    Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                      Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>        Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                               Upsert features (implies 'append
↪') [not available in pipelines]
                                                         Mutually exclusive with --append
```

Description

gdal vector simplify simplifies geometries, ensuring that the result is a valid geometry having the same dimension and number of components as the input.

The simplification uses a maximum distance difference algorithm similar to the one used in the Douglas-Peucker algorithm.

This done by the *OGRGeometry::SimplifyPreserveTopology()* method which preserves the topology per feature, but not for a whole layer. Thus gaps or overlaps between geometries that were initially contiguous may happen. To perform simplification that preserves shared boundaries between geometries, see *gdal vector simplify-coverage*.

This command can also be used as a step of *gdal vector pipeline*.

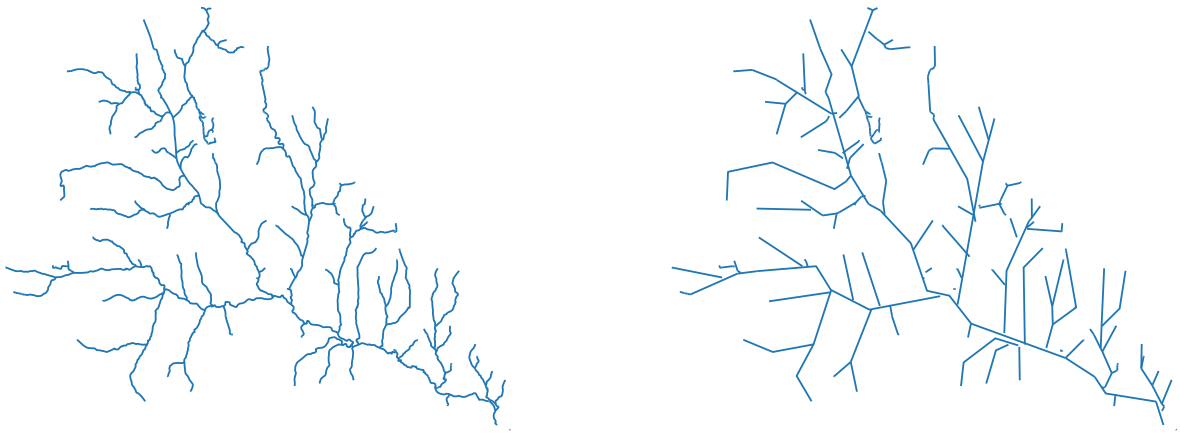


Fig. 15: Line dataset before (left) and after (right) simplification with **gdal vector simplify**.

> **Note**
>
> This command requires a GDAL build against the GEOS library.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

--tolerance <TOLERANCE>

The specified value of this option is the tolerance used to merge consecutive points of the output geometry using the *OGRGeometry::SimplifyPreserveTopology()* method The unit of the distance is in georeferenced units of the source layer.

Standard Options

--active-geometry <ACTIVE-GEOMETRY>

Set the active geometry field from its name. When it is specified, only the specified geometry field will be subject to the processing. Other geometry fields will be not modified. If this option is not specified, all geometry fields will be subject to the processing. This option can be combined together with *--active-layer*.

--active-layer <ACTIVE-LAYER>

Set the active layer. When it is specified, only the layer specified by its name will be subject to the processing. Other layers will be not modified. If this option is not specified, all layers will be subject to the processing.

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--input-layer <INPUT-LAYER>

Specifies the name of one or more layers to process. By default, all layers will be processed. To read and write all layers but only process a subset, use *--active-layer*.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-layer <NAME>

Specifies the name of the layer to which features will be written. If not specified, output layer names will be the same as input layer names.

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Simplify geometries using a tolerance of one km (assuming the CRS is in meters)

```
$ gdal vector simplify --tolerance=1000 in.gpkg out.gpkg --overwrite
```

5.1.4.37 gdal vector simplify-coverage

Added in version 3.12.

Synopsis

```
Usage: gdal vector simplify-coverage [OPTIONS] <INPUT> <OUTPUT> <TOLERANCE>

Simplify shared boundaries of a polygonal vector dataset.

Positional arguments:
  -i, --input <INPUT>                                    Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]
  --tolerance <TOLERANCE>                                Distance tolerance for␣
↪simplification. [required]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>               Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>           Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                               Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                      Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                               Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                         Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                          Output layer name [not available␣
↪in pipelines]
  --skip-errors                                          Skip errors when writing features␣
↪[not available in pipelines]
```

(continues on next page)

(continued from previous page)

```
  --active-layer <ACTIVE-LAYER>                          Set active layer (if not␣
↪specified, all)
  --preserve-boundary                                    Whether the exterior boundary␣
↪should be preserved.

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                    Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                      Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>        Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                               Upsert features (implies 'append
↪') [not available in pipelines]
                                                         Mutually exclusive with --append
```

Description

gdal vector simplify-coverage simplifies boundaries of a polygonal dataset, such that shared boundaries are preserved without introducing gaps or overlaps between features. Gaps or overlaps already present in the input dataset will not be corrected.

This requires loading the entire dataset into memory at once. If preservation of shared boundaries is not needed, *gdal vector simplify* provides an alternative that can process geometries in a streaming manner.

Simplification is performed using the Visvalingam-Whyatt algorithm.

This command can also be used as a step of *gdal vector pipeline*.

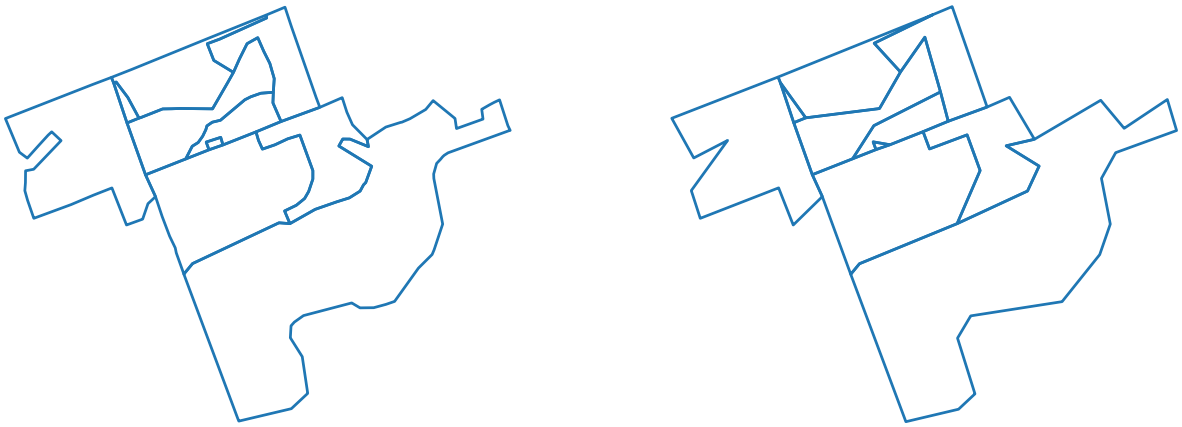


Fig. 16: Polygon dataset before (left) and after (right) simplification with **gdal vector simplify-coverage**.

> **Note**
>
> This command requires a GDAL build against the GEOS library (version 3.12 or greater).

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

> **Note**
>
> However this algorithm is not natively streaming compatible. Consequently a in-memory temporary dataset will be generated, which may cause significant processing time at opening.

Program-Specific Options

--preserve-boundary

Flag indicating whether to preserve (avoid simplifying) external boundaries. This can be useful when simplifying a portion of a larger dataset.

--tolerance <TOLERANCE>

Tolerance used for determining whether vertices should be removed. Specified in georeferenced units of the source layer.

Standard Options

--active-layer <ACTIVE-LAYER>

Set the active layer. When it is specified, only the layer specified by its name will be subject to the processing. Other layers will be not modified. If this option is not specified, all layers will be subject to the processing.

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--input-layer <INPUT-LAYER>

Specifies the name of one or more layers to process. By default, all layers will be processed. To read and write all layers but only process a subset, use `--active-layer`.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-layer <NAME>

Specifies the name of the layer to which features will be written. If not specified, output layer names will be the same as input layer names.

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of `--append` where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Simplify geometries using a tolerance of one square meter (assuming the CRS is in meters)

```
$ gdal vector simplify-coverage --tolerance=1 in.gpkg out.gpkg --overwrite
```

Example 2: Using simplify-coverage as part of a vector pipeline

This can be used to make the output geometries have a consistent type.

```
$ gdal vector pipeline ! read tl_2024_us_state.shp ! simplify-coverage --tolerance 2 !␣
↪set-geom-type --multi ! write out.gpkg --overwrite
```

5.1.4.38 gdal vector sql

Added in version 3.11.

Synopsis

```
Usage: gdal vector sql [OPTIONS] <INPUT> [<OUTPUT>]

Apply SQL statement(s) to a dataset.

Positional arguments:
  -i, --dataset, --input <INPUT>                      Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                               Output vector dataset [not␣
↪available in pipelines]

Common Options:
  -h, --help                                          Display help message and exit
  --json-usage                                        Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                              Configuration option [may be␣
↪repeated]
  -q, --quiet                                         Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
```

(continues on next page)

(continued from previous page)

```
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>         Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                          Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                             Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                    Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                             Whether appending to existing␣
↪layer is allowed [not available in pipelines]
                                                       Mutually exclusive with --upsert
  --skip-errors                                        Skip errors when writing features␣
↪[not available in pipelines]
  --sql <statement>|@<filename>                        SQL statement(s) [may be␣
↪repeated] [required]
  --output-layer <OUTPUT-LAYER>                        Output layer name(s) [may be␣
↪repeated]
  --dialect <DIALECT>                                  SQL dialect (e.g. OGRSQL, SQLITE)

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                  Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                    Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>      Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                             Upsert features (implies 'append
↪') [not available in pipelines]
                                                       Mutually exclusive with --append
```

Description

gdal vector sql returns one or several layers evaluated from SQL statements.

Starting with GDAL 3.12, when using `--update`, and without an output dataset specified, this can be used to execute statements that modify the input dataset, such as UPDATE, DELETE, etc.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Program-Specific Options

--output-layer <OUTPUT-LAYER>

Output SQL layer name(s). If not specified, a generic layer name such as "SELECT" may be generated.

Must be specified as many times as there are SQL statements, either as several --output-layer arguments, or a single one with the layer names combined with comma.

--quiet

Added in version 3.12.

Silence potential information messages.

--sql <sql_statement>|@<filename>

SQL statement to execute that returns a table/layer (typically a SELECT statement).

Can be repeated to generated multiple output layers (repeating --sql <value> for each output layer)

--dialect <DIALECT>

SQL dialect.

By default the native SQL of an RDBMS is used when using `gdal vector sql`. If using `sql` as a step of `gdal vector pipeline`, this is only true if the step preceding `sql` is `read`, otherwise the *OGRSQL* dialect is used.

If a datasource does not support SQL natively, the default is to use the `OGRSQL` dialect, which can also be specified with any data source.

The *SQL SQLite dialect* dialect can be chosen with the `SQLITE` and `INDIRECT_SQLITE` dialect values, and this can be used with any data source. Overriding the default dialect may be beneficial because the capabilities of the SQL dialects vary.

Supported dialects can be checked with `gdal --format`. For example:

```
$ gdal --format "PostgreSQL"
[...]
Supported SQL dialects: NATIVE OGRSQL SQLITE
[...]

$ gdal --format "ESRI Shapefile"
[...]
Supported SQL dialects: OGRSQL SQLITE
[...]
```

Standard Options

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

-q, --quiet

Suppress progress bar and some warning messages.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Generate a GeoPackage file with a layer sorted by descending population

```
$ gdal vector sql in.gpkg out.gpkg --output-layer country_sorted_by_pop --sql="SELECT *␣
↪FROM country ORDER BY pop DESC"
```

Example 2: Generate a GeoPackage file with 2 SQL result layers

```
$ gdal vector sql in.gpkg out.gpkg --output-layer=beginning,end --sql="SELECT * FROM my_
↪layer LIMIT 100" --sql="SELECT * FROM my_layer OFFSET 100000 LIMIT 100"
```

Example 3: Modify in-place a GeoPackage dataset

```
$ gdal vector sql --update my.gpkg --sql "DELETE FROM countries WHERE pop > 1e6"
```

Example 4: Add a new field to an existing layer of a GeoPackage

```
$ gdal vector sql --update my.gpkg --sql "ALTER TABLE countries ADD COLUMN abbrev␣
↪STRING(10)"
```

Example 5: Append to an existing layer of a GeoPackage file

```
$ gdal vector pipeline read europe.gpkg ! \
                       sql --sql "SELECT * FROM country WHERE pop > 1e6" ! \
                       write --append --output-layer-name=world world.gpkg
```

Example 6: List unique values in a Shapefile field

Bash

PowerShell

```
gdal vector pipeline \
    ! read in.shp \
    ! sql --sql "SELECT DISTINCT CODE FROM in ORDER BY CODE" \
    ! info --features
```

```
gdal vector pipeline `
    ! read in.shp `
    ! sql --sql "SELECT DISTINCT CODE FROM in ORDER BY CODE" `
    ! info --features
```

5.1.4.39 gdal vector swap-xy

Added in version 3.12.

Synopsis

```
Usage: gdal vector swap-xy [OPTIONS] <INPUT> <OUTPUT>

Swap X and Y coordinates of geometries of a vector dataset.

Positional arguments:
  -i, --input <INPUT>                                    Input vector datasets [required]␣
↪[not available in pipelines]
  -o, --output <OUTPUT>                                  Output vector dataset [required]␣
↪[not available in pipelines]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -l, --layer, --input-layer <INPUT-LAYER>               Input layer name(s) [may be␣
↪repeated] [not available in pipelines]
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)␣
↪[not available in pipelines]
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]␣
↪[not available in pipelines]
  --lco, --layer-creation-option <KEY>=<VALUE>           Layer creation option [may be␣
↪repeated] [not available in pipelines]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed [not available in pipelines]
  --update                                               Whether to open existing dataset␣
↪in update mode [not available in pipelines]
  --overwrite-layer                                      Whether overwriting existing␣
↪output layer is allowed [not available in pipelines]
  --append                                               Whether appending to existing␣
↪layer is allowed [not available in pipelines]
```

(continues on next page)

(continued from previous page)

```
                                                        Mutually exclusive with --upsert
  --output-layer <OUTPUT-LAYER>                         Output layer name [not available␣
↪in pipelines]
  --skip-errors                                         Skip errors when writing features␣
↪[not available in pipelines]
  --active-layer <ACTIVE-LAYER>                         Set active layer (if not␣
↪specified, all)
  --active-geometry <ACTIVE-GEOMETRY>                   Geometry field name to which to␣
↪restrict the processing (if not specified, all)

Advanced Options:
  --if, --input-format <INPUT-FORMAT>                   Input formats [may be repeated]␣
↪[not available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                     Open options [may be repeated]␣
↪[not available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>       Output open options [may be␣
↪repeated] [not available in pipelines]
  --upsert                                              Upsert features (implies 'append
↪') [not available in pipelines]
                                                        Mutually exclusive with --append
```

Description

gdal vector swap-xy swaps the X and Y coordinates of geometries, typically to work around axis order issues of coordinate reference systems.

It can also be used as a step of *gdal vector pipeline*.

GDALG output (on-the-fly / streamed dataset)

This program supports serializing the command line as a JSON file using the `GDALG` output format. The resulting file can then be opened as a vector dataset using the *GDALG: GDAL Streamed Algorithm* driver, and apply the specified pipeline in a on-the-fly / streamed way.

Standard Options

--active-geometry <ACTIVE-GEOMETRY>

Set the active geometry field from its name. When it is specified, only the specified geometry field will be subject to the processing. Other geometry fields will be not modified. If this option is not specified, all geometry fields will be subject to the processing. This option can be combined together with `--active-layer`.

--active-layer <ACTIVE-LAYER>

Set the active layer. When it is specified, only the layer specified by its name will be subject to the processing. Other layers will be not modified. If this option is not specified, all layers will be subject to the processing.

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--input-layer <INPUT-LAYER>

Specifies the name of one or more layers to process. By default, all layers will be processed. To read and write all layers but only process a subset, use `--active-layer`.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--output-layer <NAME>

Specifies the name of the layer to which features will be written. If not specified, output layer names will be the same as input layer names.

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--update

Whether to open an existing output dataset in update mode.

--upsert

Added in version 3.12.

Variant of *--append* where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Basic usage

```
$ gdal vector swap-xy in.gpkg out.gpkg --overwrite
```

5.1.4.40 gdal vector update

Added in version 3.13.

Synopsis

```
Usage: gdal vector update [OPTIONS] <INPUT> <OUTPUT>

Update an existing vector dataset with an input vector dataset.

Positional arguments:
  -i, --input <INPUT>                             Input vector datasets [required] [not␣
↪available in pipelines]
  -o, --output <OUTPUT>                           Output vector dataset [required]

Common Options:
  -h, --help                                      Display help message and exit
```

(continues on next page)

(continued from previous page)

```
  --json-usage                                      Display usage as JSON document and␣
↪exit
  --config <KEY>=<VALUE>                            Configuration option [may be repeated]
  -q, --quiet                                       Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  --input-layer <INPUT-LAYER>                       Input layer name
  --output-layer <OUTPUT-LAYER>                     Output layer name
  --mode <MODE>                                     Set update mode. MODE=merge|update-
↪only|append-only (default: merge)
  --key <KEY>                                       Field(s) used as a key to identify␣
↪features [may be repeated]

Advanced Options:
  --if, --input-format <INPUT-FORMAT>               Input formats [may be repeated] [not␣
↪available in pipelines]
  --oo, --open-option <KEY>=<VALUE>                 Open options [may be repeated] [not␣
↪available in pipelines]
  --output-oo, --output-open-option <KEY>=<VALUE>  Output open options [may be repeated]
```

Description

gdal vector update can be used to update the features of an existing output vector dataset with the features of the input vector. It identifies matching features between input and output based on their feature ID by default, or based on the value of one or several columns forming a key, when using *--key*.

With the default `merge` mode, features found in both layers will be updated, and features only existing in the input layer will be created in the output layer.

The schemas of the input and output layers do not need to be identical (except on columns specified by *--key*), but they should use the same CRS for geometry columns, as no on-the-fly reprojection is done. If a column exists in the input layer but not in the output layer, it will not be created. And if a column exists in the output layer and not in the input layer, its content will not be updated.

gdal vector update can be used as a step of a pipeline, with the input dataset being the output of the previous step.

Program-Specific Options

--input-layer `<INPUT-LAYER>`

Input layer name. Must be specified if the input dataset has several layers.

--key `<KEY>`

Field(s) used as a key to identify features. For multiple fields, specify *--key* multiple times.

--mode `merge|update-only|append-only`

Defines how updates are done. Default is `merge`.

When using the default `merge` mode, features found in both layers will be updated, and features only existing in the input layer will be created in the output layer.

When using the `update-only` mode, features found in both layers will be updated, and features only found in the input layer will be ignored.

When using the `append-only` mode, features found in both layers will *not* be updated, and features only found in the input layer will be created in the output layer.

--output-layer <OUTPUT-LAYER>

Output layer name. Must be specified if the output dataset has several layers.

Standard Options

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Update existing out.gpkg with content of in.gpkg, using identifier as the key.

```
$ gdal vector update --key identifier in.gpkg out.gpkg
```

5.1.4.41 gdal vector pipeline write

Added in version 3.11.

Description

The `write` operation is for use in a *gdal pipeline* only, and writes a vector dataset. This is the last step of a pipeline.

To write a temporary dataset in the middle of a pipeline, use *gdal vector materialize*.

Synopsis

```
* write [OPTIONS] <OUTPUT>
--------------------------

Write a vector dataset.

Positional arguments:
  -o, --output <OUTPUT>                                Output vector dataset [required]
```

(continues on next page)

(continued from previous page)

```
Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format ("GDALG" allowed)
  --co, --creation-option <KEY>=<VALUE>                Creation option [may be repeated]
  --lco, --layer-creation-option <KEY>=<VALUE>         Layer creation option [may be␣
↪repeated]
  --overwrite                                          Whether overwriting existing␣
↪output dataset is allowed
  --update                                             Whether to open existing dataset␣
↪in update mode
  --overwrite-layer                                    Whether overwriting existing␣
↪output layer is allowed
  --append                                             Whether appending to existing␣
↪layer is allowed
                                                       Mutually exclusive with --upsert
  -l, --output-layer <OUTPUT-LAYER>                    Output layer name
  --skip-errors                                        Skip errors when writing features
  --no-create-empty-layers                             Avoid creating layers to which no␣
↪features will be written

Advanced Options:
  --output-oo, --output-open-option <KEY>=<VALUE>      Output open options [may be␣
↪repeated]
  --upsert                                             Upsert features (implies 'append')
                                                       Mutually exclusive with --append
```

Standard Options

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional dataset creation options that can be used to control particulars about the file created. For instance, the GeoPackage driver supports creation options to control the version.

May be repeated.

The dataset creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that dataset creation options are different from layer creation options.

--lco, --layer-creation-option <NAME>=<VALUE>

Many formats have one or more optional layer creation options that can be used to control particulars about the layer created. For instance, the GeoPackage driver supports layer creation options to control the feature identifier or geometry column name, setting the identifier or description, etc.

May be repeated.

The layer creation options available vary by format driver, and some simple formats have no layer creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the

documentation for the format is the definitive source of information on driver creation options. See *Vector drivers* format specific documentation for legal creation options for each format.

Note that layer creation options are different from dataset creation options.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--update

Whether to open an existing output dataset in update mode.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--append

Whether appending features to existing layer(s) is allowed. This also creates the output dataset if it does not exist yet.

--output-layer <NAME>

Specifies the name of the layer to which features will be written. If not specified, output layer names will be the same as input layer names.

--output-open-option, --output-oo <NAME>=<VALUE>

Added in version 3.12.

Dataset open option for output dataset (format specific).

May be repeated.

--skip-errors

Added in version 3.12.

Whether failures to write feature(s) should be ignored. Note that this option sets the size of the transaction unit to one feature at a time, which may cause severe slowdown when inserting into databases.

--upsert

Added in version 3.12.

Variant of `--append` where the *OGRLayer::UpsertFeature()* operation is used to insert or update features instead of appending with *OGRLayer::CreateFeature()*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *GDAL_DCAP_UPSERT* capability).

The upsert operation uses the FID of the input feature, when it is set (and the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (the `--unset-fid` option of *gdal vector edit* can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

Examples

Example 1: Write a GeoPackage file

```
$ gdal vector pipeline ... [other commands here] ... ! write out.gpkg --overwrite
```

5.1.5 Multidimensional raster commands

5.1.5.1 gdal mdim

Added in version 3.11.

Synopsis

```
Usage: gdal mdim <SUBCOMMAND> [OPTIONS]
where <SUBCOMMAND> is one of:
  - convert: Convert a multidimensional dataset.
  - info:    Return information on a multidimensional dataset.
  - mosaic:  Build a mosaic, either virtual (VRT) or materialized, from multidimensional␣
↪datasets.

Common Options:
  -h, --help              Display help message and exit
  --json-usage            Display usage as JSON document and exit
  --config <KEY>=<VALUE>  Configuration option [may be repeated]

Options:
  --drivers               Display multidimensional driver list as JSON document
```

Available sub-commands

- *gdal mdim info*
- *gdal mdim convert*
- *gdal mdim mosaic*

Examples

Example 1: Getting information on the file temperatures.nc (with JSON output)

```
$ gdal mdim info temperatures.nc
```

Example 2: Converting file temperatures.nc to Zarr

```
$ gdal mdim convert temperatures.nc temperatures.zarr
```

Example 3: Getting the list of multidimensional drivers (with JSON output)

```
$ gdal mdim --drivers
```

5.1.5.2 gdal mdim info

Added in version 3.11.

Synopsis

```
Usage: gdal mdim info [OPTIONS] <INPUT>

Return information on a multidimensional dataset.

Positional arguments:
  -i, --dataset, --input <INPUT>       Input multidimensional raster dataset [required]

Common Options:
  -h, --help                           Display help message and exit
  --json-usage                         Display usage as JSON document and exit
  --config <KEY>=<VALUE>               Configuration option [may be repeated]

Options:
  --summary                            Report only group and array hierarchy, without␣
↪detailed information on attributes or dimensions.
                                       Mutually exclusive with --detailed
  --detailed                           Most verbose output. Report attribute data types␣
↪and array values.
                                       Mutually exclusive with --summary
  --array <ARRAY>                      Name of the array, used to restrict the output to␣
↪the specified array.
  --limit <LIMIT>                      Number of values in each dimension that is used␣
↪to limit the display of array values.
  --array-option <KEY>=<VALUE>         Option passed to GDALGroup::GetMDArrayNames() to␣
↪filter reported arrays. [may be repeated]
  --stats                              Read and display image statistics.

Advanced Options:
  --oo, --open-option <KEY>=<VALUE>    Open options [may be repeated]
  --if, --input-format <INPUT-FORMAT>  Input formats [may be repeated]
```

Description

gdal mdim info lists various information about a GDAL supported multidimensional dataset, and returns them on the standard output stream when used from the command line, or in the `output` parameter when used from the API.

The following items will be reported (when known) as JSON:

- The format driver used to access the file.
- Hierarchy of groups
- Group attributes
- **Arrays:**
  - Name
  - Dimension name, sizes, indexing variable
  - Data type
  - Attributes
  - SRS
  - Nodata value

- Units
- Statistics (if requested)

The following options are available:

Standard options

--summary

Added in version 3.13.

Report only group and array hierarchy, without detailed information on attributes or dimensions. Mutually exclusive with *--detailed*.

--detailed

Most verbose output. Report attribute data types and array values. Mutually exclusive with *--summary*.

--array <array_name>

Name of the array used to restrict the output to the specified array.

> **Note**
>
> Bash completion for *--array* is possible if the dataset name is specified before.

--limit <number>

Number of values in each dimension that is used to limit the display of array values. By default, unlimited. Only taken into account if used with -detailed.

--array-option <NAME>=<VALUE>

Option passed to *GDALGroup::GetMDArrayNames()* to filter reported arrays. Such option is format specific. Consult driver documentation. This option may be used several times.

> **Note**
>
> Bash completion for *--array-option* is possible if the dataset name is specified before.

--stats

Read and display array statistics. Force computation if no statistics are stored in an array.

Advanced options

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Getting information on the file netcdf-4d.nc as JSON output

```
$ gdal mdim info netcdf-4d.nc
```

```
{
  "type": "group",
  "name": "/",
  "attributes": {
    "Conventions": "CF-1.5"
  },
  "dimensions": [
    {
      "name": "levelist",
      "full_name": "/levelist",
      "size": 2,
      "type": "VERTICAL",
      "indexing_variable": "/levelist"
    },
    {
      "name": "longitude",
      "full_name": "/longitude",
      "size": 10,
      "type": "HORIZONTAL_X",
      "direction": "EAST",
      "indexing_variable": "/longitude"
    },
    {
      "name": "latitude",
      "full_name": "/latitude",
      "size": 10,
      "type": "HORIZONTAL_Y",
      "direction": "NORTH",
      "indexing_variable": "/latitude"
    },
    {
      "name": "time",
      "full_name": "/time",
        "size": 4,
      "type": "TEMPORAL",
      "indexing_variable": "/time"
      }
  ],
  "arrays": {
    "levelist": {
      "datatype": "Int32",
      "dimensions": [
```

(continues on next page)

(continued from previous page)

```
          "/levelist"
        ],
      "attributes": {
        "long_name": "pressure_level"
      },
      "unit": "millibars"
    },
    "longitude": {
      "datatype": "Float32",
      "dimensions": [
        "/longitude"
      ],
      "attributes": {
        "standard_name": "longitude",
        "long_name": "longitude",
        "axis": "X"
      },
      "unit": "degrees_east"
    },
    "latitude": {
      "datatype": "Float32",
      "dimensions": [
        "/latitude"
      ],
      "attributes": {
        "standard_name": "latitude",
        "long_name": "latitude",
        "axis": "Y"
      },
      "unit": "degrees_north"
    },
    "time": {
      "datatype": "Float64",
      "dimensions": [
        "/time"
      ],
      "attributes": {
        "standard_name": "time",
        "calendar": "standard"
      },
      "unit": "hours since 1900-01-01 00:00:00"
    },
    "t": {
      "datatype": "Int32",
      "dimensions": [
        "/time",
        "/levelist",
        "/latitude",
        "/longitude"
      ],
      "nodata_value": -32767
    }
```

(continues on next page)

(continued from previous page)

```
  },
  "structural_info": {
    "NC_FORMAT": "CLASSIC"
  }
}
```

5.1.5.3 gdal mdim convert

Added in version 3.11.

Synopsis

```
Usage: gdal mdim convert [OPTIONS] <INPUT> <OUTPUT>

Convert a multidimensional dataset.

Positional arguments:
  -i, --input <INPUT>                                    Input raster or multidimensional␣
↪raster dataset [required]
  -o, --output <OUTPUT>                                  Output multidimensional raster␣
↪dataset (created by algorithm) [required]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]
  -q, --quiet                                            Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format
  --co, --creation-option <KEY>=<VALUE>                  Creation option [may be repeated]
  --overwrite                                            Whether overwriting existing␣
↪output dataset is allowed
  --array <ARRAY-SPEC>                                   Select a single array instead of␣
↪converting the whole dataset. [may be repeated]
  --array-option <KEY>=<VALUE>                           Option passed to␣
↪GDALGroup::GetMDArrayNames() to filter arrays. [may be repeated]
  --group <GROUP-SPEC>                                   Select a single group instead of␣
↪converting the whole dataset. [may be repeated]
  --subset <SUBSET-SPEC>                                 Select a subset of the data. [may␣
↪be repeated]
  --scale-axes <SCALEAXES-SPEC>                          Applies a integral scale factor␣
↪to one or several dimensions [may be repeated]
  --strict                                               Turn warnings into failures.

Advanced Options:
  --oo, --open-option <KEY>=<VALUE>                      Open options [may be repeated]
  --if, --input-format <INPUT-FORMAT>                    Input formats [may be repeated]
```

Description

gdal mdim convert can be used to convert multidimensional data between different formats.

The following options are available:

Standard options

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Select the output format. This can be a format that supports multidimensional output (such as *NetCDF: Network Common Data Form*, *Multidimensional VRT*), or a "classic" 2D formats, if only one single 2D array results of the other specified conversion operations. When this option is not specified, the format is guessed when possible from the extension of the destination filename.

--co, --creation-option <KEY>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--format* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

Array-level creation options may be passed by prefixing them with `ARRAY:`. See *`GDALGroup::CopyFrom()`* for further details regarding such options.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--array <array_spec>

Instead of converting the whole dataset, select one array, and possibly perform operations on it. This option can be specified several times to operate on different arrays.

<array_spec> may be just an array name, potentially using a fully qualified syntax (/group/subgroup/array_name). Or it can be a combination of options with the syntax: name={src_array_name}[,dstname={dst_array_name}][,resample=yes][,transpose=[{axis1},{axis2},...][,view={view_expr}]

The following options are processed in that order:

- `resample=yes` asks for the array to run through *`GDALMDArray::GetResampled()`*.
- **[{axis1},{axis2},...] is the argument of *`GDALMDArray::Transpose()`*.**
  For example, transpose=[1,0] switches the axis order of a 2D array. See *Example 6*.
- {view_expr} is the value of the *viewExpr* argument of *`GDALMDArray::GetView()`*. See *Example 5*.

When specifying a view_expr that performs a slicing or subsetting on a dimension, the equivalent operation will be applied to the corresponding indexing variable.

--array-option <NAME>=<VALUE>

Option passed to *`GDALGroup::GetMDArrayNames()`* to filter reported arrays. Such option is format specific. Consult driver documentation. This option may be used several times.

--group <group_spec>

Instead of converting the whole dataset, select one group, and possibly perform operations on it. This option can be specified several times to operate on different groups. If only one group is specified, its content will be copied directly to the target root group. If several ones are specified, they are copied under the target root group

<group_spec> may be just a group name, potentially using a fully qualified syntax (/group/subgroup/subsubgroup_name). Or it can be a combination of options with the syntax: `name={src_group_name}[,dstname={dst_group_name}][,recursive=no]`

--subset <subset_spec>

Performs a subsetting (trimming or slicing) operation along a dimension, provided that it is indexed by a 1D variable of numeric or string data type, and whose values are monotonically sorted. <subset_spec> follows exactly the OGC WCS 2.0 KVP encoding for subsetting.

That is `dim_name(min_val,max_val)` or `dim_name(sliced_val)` The first syntax will subset the dimension `dim_name` to values in the [min_val,max_val] range. The second syntax will slice the dimension dim_name to value sliced_val (and this dimension will be removed from the arrays that reference to it)

Beware that dimension name is case sensitive.

Several subsetting operations along different dimensions can be specified by repeating the option.

Using --subset is incompatible with specifying a *view* option in --array.

See *Example 2* or *Example 3*.

--scale-axes <scaleaxes_spec>

Applies a integral scale factor to one or several dimensions, that is extract 1 value every N values (without resampling).

<scaleaxes_spec> follows exactly the syntax of the KVP encoding of the SCALEAXES parameter of OGC WCS 2.0 Scaling Extension, but limited to integer scale factors.

That is `<dim1_name>(<scale_factor>)`.

Several scaling operations along different dimensions can be specified by repeating the option.

Using --scale-axes is incompatible with specifying a *view* option in --array.

See *Example 4*.

--strict

By default, some failures during the translation are tolerated, such as not being able to write group attributes. When setting this option, such failures will cause the process to fail.

Advanced options

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Convert a netCDF file to a multidimensional VRT file

```
gdal mdim convert in.nc out.vrt
```

Example 2: Extract a 2D slice of a time,Y,X array

```
gdal mdim convert in.nc out.tif --subset "time(\"2010-01-01\")" --array temperature
```

Example 3: Extract a a 3D chunk from a time-indexed GRIB file into a multiband GeoTIFF.

```
gdal mdim convert /vsicurl/https://tgftp.nws.noaa.gov/SL.us008001/ST.opnl/DF.gr2/DC.ndfd/
↪AR.conus/VP.001-003/ds.qpf.bin tst.tif --subset "TIME(1763164800,1763251200)" --array␣
↪QPF_0-SFC --co COMPRESS=DEFLATE
```

Example 4: Subsample along x and y axis

```
gdal mdim convert in.nc out.nc --scale-axes "x(2),y(2)"
```

Example 5: Reorder the values of an array

Reorder the values of the time,Y,X array along the Y axis from top-to-bottom to bottom-to-top (or the reverse)

```
gdal mdim convert in.nc out.nc --array "name=temperature,view=[:,::-1,:]"
```

Example 6: Transpose an array that has X,Y,time dimension order to time,Y,X

```
gdal mdim convert in.nc out.nc --array "name=temperature,transpose=[2,1,0]"
```

5.1.5.4 gdal mdim mosaic

Added in version 3.12.

Synopsis

```
Usage: gdal mdim mosaic [OPTIONS] <INPUT>... <OUTPUT>

Build a mosaic, either virtual (VRT) or materialized, from multidimensional datasets.

Positional arguments:
  -i, --input <INPUT>                                    Input multidimensional raster␣
↪datasets [1.. values] [required]
  -o, --output <OUTPUT>                                  Output multidimensional raster␣
↪dataset (created by algorithm) [required]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
```

(continues on next page)

(continued from previous page)

```
↪and exit
  --config <KEY>=<VALUE>                                Configuration option [may be␣
↪repeated]
  -q, --quiet                                           Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format
  --co, --creation-option <KEY>=<VALUE>                 Creation option [may be repeated]
  --overwrite                                           Whether overwriting existing␣
↪output dataset is allowed
  --array <ARRAY>                                       Name of array(s) to mosaic. [may␣
↪be repeated]

Advanced Options:
  --oo, --open-option <KEY>=<VALUE>                     Open options [may be repeated]
  --if, --input-format <INPUT-FORMAT>                   Input formats [may be repeated]
```

Description

gdal mdim mosaic builds a mosaic of multidimensional array(s), from a list of input GDAL multidimensional datasets, that can be either a virtual mosaic in the *Multidimensional VRT (Virtual Dataset)* format, or in a more conventional multidimensional format such as *NetCDF: Network Common Data Form* or *Zarr*.

Wildcards '*', '?' or '[']of *VSIGlob()* can be used for input dataset names, even on files located on network file systems such as /vsis3/, /vsigs/, /vsiaz/, etc. Alternatively if an input dataset name is prefixed by the @ character, it will be assumed to contain the list of actual dataset names (one per line) to use as input datasets.

To create a mosaic from an array, certain structural constraints apply to all contributing arrays from the input datasets:

- All contributing arrays must have the same array name and path (e.g. "/measurements/pressure")
- They must have the same data type and nodata value
- They must share the same number of dimensions, in the same order, and with the same names.
- each dimension of the array must have an associated one-dimensional numeric indexing variable.

A typical use case is when a multidimensional array is sliced into several files, and it is desired to reconstruct the full array.

The program can support mosaicing along several axes.

There might be "holes" among the source datasets along the mosaiced dimensions, and possibly overlaps (but only when regularly spacing along a mosaiced dimension). Holes are filled with the nodata value, if defined in the source arrays, or zero-initialized otherwise. In case of overlap, the order of the input list is used to determine priority. Files that are listed at the end are the ones from which the content will be fetched.

Different typical use cases that can be addressed by this program are:

- a 3D array is sliced along a single dimension (for example a temporal one) into several files with each slice having a sample along that dimension. In that use case, the timestamps of the slices do not need to be regularly spaced.
- a 2D array is chunked/tiled along X and Y dimensions, with a regular spacing along those X and Y dimensions, and where the coordinate values are aligned on the same grid among chunks.

By default, a target array is created for each array with two or more dimensions. This behavior can be restricted to specific arrays using the *--array* option.

The following options are available:

Standard options

-i, --input <INPUT>

Input multidimensional dataset names. Required.

Wildcards '*', '?' or '['] of *VSIGlob()* can be used for input dataset names, even on files located on network file systems such as /vsis3/, /vsigs/, /vsiaz/, etc. Alternatively if a input dataset name is prefixed by the @ character, it will be assumed to contain the list of actual dataset names (one per line) to use as input datasets.

-o, --output <OUTPUT>

Output multidimensional dataset name. Required.

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Select the output format. This can be a format that supports multidimensional output (such as *NetCDF: Network Common Data Form*, *Multidimensional VRT*). When this option is not specified, the format is guessed when possible from the extension of the destination filename.

--co, --creation-option <KEY>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--format* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

Array-level creation options may be passed by prefixing them with `ARRAY:`. See *GDALGroup::CopyFrom()* for further details regarding such options.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--array <ARRAYS>

Name or full path of one or several arrays to mosaic. If not specified, a target array is created for each array with two or more dimensions.

Advanced options

--oo, --open-option <NAME>=<VALUE>

Dataset open option (format specific).

May be repeated.

--if, --input-format <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

May be repeated.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Mosaic together all netCDF files starting with slice that are slices of a 3D array

```
gdal mdim mosaic slice*.nc out.vrt
```

5.1.6 Dataset management commands

5.1.6.1 gdal dataset

Added in version 3.12.

gdal dataset can perform various operations on data files, depending on the chosen operation. This includes identifying data types and deleting, renaming or copying the files.

Synopsis

```
Usage: gdal dataset <SUBCOMMAND> [OPTIONS]
where <SUBCOMMAND> is one of:
  - check:    Check whether there are errors when reading the content of a dataset.
  - copy:     Copy files of a dataset. (alias: cp)
  - delete:   Delete dataset(s). (alias: rm, remove)
  - identify: Identify driver opening dataset(s).
  - rename:   Rename files of a dataset. (alias: ren, mv)

Common Options:
  -h, --help              Display help message and exit
  --json-usage            Display usage as JSON document and exit
  --config <KEY>=<VALUE>  Configuration option [may be repeated]
```

Available sub-commands

- *gdal dataset identify*: Identify driver opening dataset(s)
- *gdal dataset check*: Check whether there are errors when reading the content of a dataset.
- *gdal dataset copy*: Copy files of a dataset.
- *gdal dataset rename*: Rename files of a dataset.
- *gdal dataset delete*: Delete dataset(s)

5.1.6.2 gdal dataset identify

Added in version 3.12.

gdal dataset identify reports the name of drivers that can open one or several dataset(s).

Synopsis

```
Usage: gdal dataset identify [OPTIONS] <FILENAME>

Identify driver opening dataset(s).

Positional arguments:
  --input, --filename <FILENAME>                      File or directory name [may be␣
↪repeated] [required]

Common Options:
  -h, --help                                          Display help message and exit
  --json-usage                                        Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                              Configuration option [may be␣
↪repeated]
  -q, --quiet                                         Quiet mode (no progress bar or␣
↪warning message) [not available in pipelines]

Options:
  -o, --output <OUTPUT>                               Output vector dataset (created by␣
↪algorithm)
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format
  --co, --creation-option <KEY>=<VALUE>               Creation option [may be repeated]
  --lco, --layer-creation-option <KEY>=<VALUE>        Layer creation option [may be␣
↪repeated]
  -l, --output-layer <OUTPUT-LAYER>                   Output layer name
  --overwrite                                         Whether overwriting existing␣
↪output dataset is allowed
  -r, --recursive                                     Recursively scan files/folders␣
↪for datasets
  --force-recursive                                   Recursively scan folders for␣
↪datasets, forcing recursion in folders recognized as valid formats
  --detailed                                          Most detailed output. Reports the␣
↪presence of georeferencing, if a GeoTIFF file is cloud optimized, etc.
  --report-failures                                   Report failures if file type is␣
↪unidentified
```

Options

--filename <FILENAME>

Any file name or directory name. Required. May be repeated

-o, --output <OUTPUT>

Added in version 3.13.

Output vector dataset that will have at a minimum fields `filename` and `driver`, and if *--detailed* is specified, also `layout` (whose value can be for example `COG`), `file_list` (side-car files), `has_crs`, `has_geotransform` and `has_overview`.

When this argument is not specified, the result report is emitted to the standard output stream, either as JSON or text (depending on *--output-format*), for command line execution, or in the `output-string` argument when used from the API.

Note that this is not a positional argument, so the `-o` or `--output` switch must be explicitly used before specifying

the name.

-f, --of, --format, --output-format `json|text|<OTHER-VECTOR-FORMAT>`

Which output format to use. Default is JSON, or text when invoked from command line.

Since GDAL 3.13, other GDAL vector formats with creation capabilities can be used, in which case *--output* must be specified.

-r, --recursive

Recursively scan files/folders for datasets.

--force-recursive

Recursively scan folders for datasets, forcing recursion in folders recognized as valid formats.

--detailed

Added in version 3.13.

Increases the level of details in the output. Reports the presence of georeferencing, if a GeoTIFF file is cloud optimized, etc.

--report-failures

Report failures if file type is unidentified.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Identifying a single file

```
$ gdal dataset identify NE1_50M_SR_W.tif

NE1_50M_SR_W.tif: GTiff
```

Example 2: Identifying a single file with JSON output

```
$ gdal dataset identify --of=JSON NE1_50M_SR_W.tif
```

```
[
   {
     "name": "NE1_50M_SR_W.tif",
     "driver": "GTiff"
   }
]
```

Example 3: Recursive mode will scan subfolders and report the data format

```
$ gdal dataset identify -r 50m_raster/

NE1_50M_SR_W/ne1_50m.jpg: JPEG
NE1_50M_SR_W/ne1_50m.png: PNG
NE1_50M_SR_W/ne1_50m_20pct.tif: GTiff
```

(continues on next page)

(continued from previous page)

```
NE1_50M_SR_W/ne1_50m_band1.tif: GTiff
NE1_50M_SR_W/ne1_50m_print.png: PNG
NE1_50M_SR_W/NE1_50M_SR_W.aux: HFA
NE1_50M_SR_W/NE1_50M_SR_W.tif: GTiff
NE1_50M_SR_W/ne1_50m_sub.tif: GTiff
NE1_50M_SR_W/ne1_50m_sub2.tif: GTiff
```

Example 4: Recursively scans subfolders and reports detailed information into a CSV file

```
$ gdal dataset identify --output out.csv --detailed -r 50m_raster/
```

Example 5: Check if a GeoTIFF file is cloud optimized using --detailed

The following returns driver, and image layout details: `GTiff, layout=COG, has CRS, has geotransform, has overview(s)`

```
$ gdal dataset identify /vsicurl/https://sentinel-cogs.s3.us-west-2.amazonaws.
↪com/sentinel-s2-l2a-cogs/36/Q/WD/2020/7/S2A_36QWD_20200701_0_L2A/TCI.tif --
↪detailed
```

5.1.6.3 `gdal dataset check`

Added in version 3.13.

`gdal dataset check` check whether there are errors when reading the content of a dataset and return 1 as the program exit code (or in the `return-code` output parameter of the algorithm) if that happens, or 0 when no errors are detected.

> **Warning**
>
> Most drivers or data formats do not have a built-in integrity mechanism. Thus a successful run of that program does not necessarily mean that the dataset is identical to the state it had just after being generated. The checks performed are generally sufficient to detect whether the dataset is truncated compared to its expected size, but this is not even necessarily guaranteed for some formats (for example the CSV driver may not be able to detect truncation).

Synopsis

```
Usage: gdal dataset check [OPTIONS] <INPUT>

Check whether there are errors when reading the content of a dataset.

Positional arguments:
  -i, --input <INPUT>                 Input raster, vector or multidimensional raster␣
↪dataset [required]

Common Options:
  -h, --help                          Display help message and exit
  --json-usage                        Display usage as JSON document and exit
  --config <KEY>=<VALUE>              Configuration option [may be repeated]
  -q, --quiet                         Quiet mode (no progress bar or warning message)␣
↪[not available in pipelines]
```

(continues on next page)

(continued from previous page)

```
Advanced Options:
  --oo, --open-option <KEY>=<VALUE>    Open options [may be repeated]
  --if, --input-format <INPUT-FORMAT>  Input formats [may be repeated]
```

Options

--input <FILENAME>

Input vector, raster or multidimensional dataset. Required.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Check whether there are errors when reading a dataset

```
$ gdal dataset check NE1_50M_SR_W.tif
```

5.1.6.4 gdal dataset copy

Added in version 3.12.

gdal dataset copy creates a copy of the dataset file(s), including potential side-car/associated files.

Synopsis

```
Usage: gdal dataset copy [OPTIONS] <SOURCE> <DESTINATION>

Copy files of a dataset.

Positional arguments:
  --source <SOURCE>            Source dataset name [required]
  --destination <DESTINATION>  Destination dataset name [required]

Common Options:
  -h, --help                   Display help message and exit
  --json-usage                 Display usage as JSON document and exit
  --config <KEY>=<VALUE>       Configuration option [may be repeated]

Options:
  --overwrite                  Whether overwriting existing output dataset is allowed

Advanced Options:
  -f, --format <FORMAT>        Dataset format
```

Options

--source <FILENAME>

Source file name or directory name. Required.

--destination <FILENAME>

Destination file name or directory name. Required.

-f, ---format <FORMAT>

Dataset format. Helps if automatic detection does not work.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Copy a dataset

```
$ gdal dataset copy source.tif destination.tif
```

5.1.6.5 gdal dataset rename

Added in version 3.12.

gdal dataset rename rename the dataset file(s), including potential side-car/associated files.

Synopsis

```
Usage: gdal dataset rename [OPTIONS] <SOURCE> <DESTINATION>

Rename files of a dataset.

Positional arguments:
  --source <SOURCE>            Source dataset name [required]
  --destination <DESTINATION>  Destination dataset name [required]

Common Options:
  -h, --help                   Display help message and exit
  --json-usage                 Display usage as JSON document and exit
  --config <KEY>=<VALUE>       Configuration option [may be repeated]

Options:
  --overwrite                  Whether overwriting existing output dataset is allowed

Advanced Options:
  -f, --format <FORMAT>        Dataset format
```

Options

--source <FILENAME>

Source file name or directory name. Required.

--destination <FILENAME>

Destination file name or directory name. Required.

-f, ---format <FORMAT>

Dataset format. Overrides the automatic format detection.

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Rename a dataset

```
$ gdal dataset rename old_name.tif new_name.tif
```

5.1.6.6 gdal dataset delete

Added in version 3.12.

gdal dataset delete delete dataset(s), including potential side-car/associated files.

Synopsis

```
Usage: gdal dataset delete [OPTIONS] <FILENAME>

Delete dataset(s).

Positional arguments:
  --filename <FILENAME>   File or directory name [may be repeated] [required]

Common Options:
  -h, --help              Display help message and exit
  --json-usage            Display usage as JSON document and exit
  --config <KEY>=<VALUE>  Configuration option [may be repeated]

Advanced Options:
  -f, --format <FORMAT>   Dataset format
```

Options

--filename <FILENAME>

Any file name or directory name. Required. May be repeated

-f, ---format <FORMAT>

Dataset format. Helps if automatic detection does not work.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Delete a dataset

```
$ gdal dataset delete NE1_50M_SR_W.tif
```

5.1.7 Virtual System Interface (VSI) commands

5.1.7.1 gdal vsi

Added in version 3.11.

The subcommands of **gdal vsi** allow manipulation of files located on the *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*.

Synopsis

```
Usage: gdal vsi <SUBCOMMAND> [OPTIONS]
where <SUBCOMMAND> is one of:
  - copy:   Copy files located on GDAL Virtual System Interface (VSI). (alias: cp)
  - delete: Delete files located on GDAL Virtual System Interface (VSI). (alias: rm,␣
↪rmdir, del)
  - list:   List files of one of the GDAL Virtual System Interface (VSI). (alias: ls)
  - move:   Move/rename a file/directory located on GDAL Virtual System Interface (VSI).␣
↪(alias: mv, ren, rename)
  - sozip:  Seek-optimized ZIP (SOZIP) commands.
  - sync:   Synchronize source and target file/directory located on GDAL Virtual System␣
↪Interface (VSI).

Common Options:
  -h, --help              Display help message and exit
  --json-usage            Display usage as JSON document and exit
  --config <KEY>=<VALUE>  Configuration option [may be repeated]
```

Available sub-commands

- *gdal vsi copy*
- *gdal vsi delete*
- *gdal vsi list*
- *gdal vsi move*
- *gdal vsi sync*
- *gdal vsi sozip*

Examples

Example 1: Listing recursively files in /vsis3/bucket with details

```
$ gdal vsi list -lR /vsis3/bucket
```

5.1.7.2 gdal vsi copy

Added in version 3.11.

Synopsis

```
Usage: gdal vsi copy [OPTIONS] <SOURCE> <DESTINATION>

Copy files located on GDAL Virtual System Interface (VSI).

Positional arguments:
  --source <SOURCE>            Source file or directory name [required]
  --destination <DESTINATION>  Destination file or directory name [required]

Common Options:
  -h, --help                   Display help message and exit
  --json-usage                 Display usage as JSON document and exit
  --config <KEY>=<VALUE>       Configuration option [may be repeated]
  -q, --quiet                  Quiet mode (no progress bar or warning message) [not␣
↪available in pipelines]

Options:
  -r, --recursive              Copy subdirectories recursively
  --skip-errors                Skip errors
```

Description

gdal vsi copy copy files and directories located on *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*.

It can copy files and directories between different virtual file systems.

This is the equivalent of the UNIX `cp` command, and `gdal vsi cp` is an alias for `gdal vsi copy`.

Options

-r, --recursive

Copy directories recursively.

--skip-errors

Skip errors that occur while while copying.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Copy recursively files from /vsis3/bucket/my_dir to local directory, creating a my_dir directory if it does not exist.

```
$ gdal vsi copy -r /vsis3/bucket/my_dir .
```

Example 2: Copy recursively files from /vsis3/bucket/my_dir to local directory, without creating a my_dir directory, without progress bar

```
$ gdal vsi copy --quiet -r /vsis3/bucket/my_dir/* .
```

5.1.7.3 gdal vsi delete

Added in version 3.11.

Synopsis

```
Usage: gdal vsi delete [OPTIONS] <FILENAME>

Delete files located on GDAL Virtual System Interface (VSI).

Positional arguments:
  --filename <FILENAME>   File or directory name to delete [required]

Common Options:
  -h, --help              Display help message and exit
  --json-usage            Display usage as JSON document and exit
  --config <KEY>=<VALUE>  Configuration option [may be repeated]

Options:
  -r, -R, --recursive     Delete directories recursively
```

Description

gdal vsi delete delete files and directories located on *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*.

This is the equivalent of the UNIX `rm` command, and `gdal vsi rm` is an alias for `gdal vsi delete`.

> **Warning**
>
> Be careful. This command cannot be undone. It can also act on the "real" file system.

Options

-r, -R, --recursive

Delete directories recursively.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Delete recursively files from /vsis3/bucket/my_dir

```
$ gdal vsi delete -r /vsis3/bucket/my_dir
```

5.1.7.4 gdal vsi list

Added in version 3.11.

Synopsis

```
Usage: gdal vsi list [OPTIONS] <FILENAME>

List files of one of the GDAL Virtual System Interface (VSI).

Positional arguments:
  --filename <FILENAME>                                File or directory name [required]

Common Options:
  -h, --help                                           Display help message and exit
  --json-usage                                         Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                               Configuration option [may be␣
↪repeated]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format. OUTPUT-
↪FORMAT=json|text
  -l, --long, --long-listing                           Use a long listing format
  -R, --recursive                                      List subdirectories recursively
  --depth <DEPTH>                                      Maximum depth in recursive mode
  --abs, --absolute-path                               Display absolute path
  --tree                                               Use a hierarchical presentation␣
↪for JSON output
```

Description

gdal vsi list list files of *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*.

This is the equivalent of the UNIX `ls` command, and `gdal vsi ls` is an alias for `gdal vsi list`.

By default, it outputs file names, at the immediate level, without details, and in JSON format.

Options

--filename <FILENAME>

Any file name or directory name, of one of the GDAL Virtual file systems. Required.

-f, --of, --format, --output-format `json|text`

Which output format to use. Default is JSON, and starting with GDAL 3.12, text when invoked from command line.

-l, --long, --long-listing

Use a long listing format, adding permissions, file size and last modification date.

-R, --recursive

List subdirectories recursively. By default the depth is unlimited, but it can be reduced with *--depth*.

--depth <DEPTH>

Maximum depth in recursive mode. 1 corresponds to no recursion, 2 to the immediate subdirectories, etc.

--abs, --absolute-path

Whether to report file names as absolute paths. By default, they are relative to the input file name.

--tree

Use a hierarchical presentation for JSON output, instead of a flat list. Only valid when *--output-format* is set to `json` (or let at its default value).

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Listing recursively files in /vsis3/bucket with details

```
$ gdal vsi list -lR /vsis3/bucket
```

5.1.7.5 gdal vsi move

Added in version 3.11.

Synopsis

```
Usage: gdal vsi move [OPTIONS] <SOURCE> <DESTINATION>

Move/rename a file/directory located on GDAL Virtual System Interface (VSI).

Positional arguments:
  --source <SOURCE>            Source file or directory name [required]
  --destination <DESTINATION>  Destination file or directory name [required]

Common Options:
  -h, --help                   Display help message and exit
  --json-usage                 Display usage as JSON document and exit
  --config <KEY>=<VALUE>       Configuration option [may be repeated]
```

(continues on next page)

(continued from previous page)

```
  -q, --quiet                  Quiet mode (no progress bar or warning message) [not␣
↪available in pipelines]
```

Description

gdal vsi move move files and directories located on *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*.

If the destination path is an existing directory, the file will be moved to it.

It can move files and directories between different virtual file systems, but this will involve copying and deletion.

Note that for cloud object storage, moving/renaming a directory may involve renaming all files it contains recursively, and is thus not an atomic operation (and could be slow and expensive on directories with many files!)

This is implemented by *VSIMove()*.

This is an analog of the UNIX `mv` command, and `gdal vsi mv` is an alias for `gdal vsi move`.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Rename a file within the same virtual file system

```
$ gdal vsi move /vsis3/bucket/my.tif /vsis3/bucket/new_name.tif
```

Example 2: Move a file into another directory within the same virtual file system

```
$ gdal vsi move /vsis3/bucket/my.tif /vsis3/bucket/existing_subdir
```

Example 3: Move a directory between two different virtual file systems

```
$ gdal vsi move /vsis3/bucket/my_directory /vsigs/bucket/
```

5.1.7.6 gdal vsi sync

Added in version 3.11.

Synopsis

```
Usage: gdal vsi sync [OPTIONS] <SOURCE> <DESTINATION>

Synchronize source and target file/directory located on GDAL Virtual System Interface␣
↪(VSI).

Positional arguments:
  --source <SOURCE>                Source file or directory name [required]
  --destination <DESTINATION>      Destination file or directory name [required]
```

(continues on next page)

(continued from previous page)

```
Common Options:
  -h, --help                        Display help message and exit
  --json-usage                      Display usage as JSON document and exit
  --config <KEY>=<VALUE>            Configuration option [may be repeated]
  -q, --quiet                       Quiet mode (no progress bar or warning message) [not␣
↪available in pipelines]

Options:
  -r, --recursive                   Synchronize recursively
  --strategy <STRATEGY>             Synchronization strategy.␣
↪STRATEGY=timestamp|ETag|overwrite (default: timestamp)
  -j, --num-threads <NUM-THREADS>   Number of jobs (or ALL_CPUS)
```

Description

gdal vsi sync synchronize files and directories located on *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*.

This is an analog to the UNIX **rsync** command. In the current implementation, **rsync** would be more efficient for local file copying, but **gdal vsi sync** main interest is when the source or target is a remote file system like /vsis3/ or /vsigs/, in which case it can take into account the timestamps of the files (or optionally the ETag/MD5Sum) to avoid unneeded copy operations.

This is implemented efficiently for:

- local filesystem <--> remote filesystem.
- remote filesystem <--> remote filesystem, where the source and target remote filesystems are the same and one of /vsis3/, /vsigs/ or /vsiaz/. Or when the target is /vsiaz/ and the source is /vsis3/, /vsigs/, /vsiadls/ or /vsicurl/

Similarly to **rsync** behavior, if the source filename ends with a slash, it means that the content of the directory must be copied, but not the directory name. For example, assuming `/home/even/foo` contains a file `bar`, `gdal vsi sync -r /home/even/foo/ /mnt/media` will create a `/mnt/media/bar` file. Whereas `gdal vsi sync -r /home/even/foo /mnt/media` will create a `/mnt/media/foo` directory which contains a `bar` file.

This is implemented by *VSISync()*.

Options

-r, --recursive

Synchronize recursively.

--strategy `timestamp|ETag|overwrite`

Determines which criterion is used to determine if a target file must be replaced when it already exists and has the same file size as the source. Only applies for a source or target being a network filesystem.

The default is `timestamp` (similarly to how **aws s3 sync** works). For an upload operation, a remote file is replaced if it has a different size or if it is older than the source. For a download operation, a local file is replaced if it has a different size or if it is newer than the remote file.

The `ETag` strategy assumes that the ETag metadata of the remote file is the MD5Sum of the file content, which is only true in the case of /vsis3/ for files not using KMS server side encryption and uploaded in a single PUT operation (so smaller than 50 MB given the default used by GDAL). Only to be used for /vsis3/, /vsigs/ or other filesystems using a MD5Sum as ETAG.

The `overwrite` strategy will always overwrite the target file with the source one.

-j, --num-threads <value>

Number of jobs to run at once

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Synchronize a local directory onto a S3 bucket

```
$ gdal vsi sync -r my_directory/ /vsis3/bucket/my_directory
```

5.1.7.7 gdal vsi sozip

Added in version 3.11.

Description

The **gdal vsi sozip** utility can be used to:

- create a *SOZip (Seek-Optimized ZIP)* file
- append files to an existing ZIP/SOZip file
- list the contents of a ZIP/SOZip file
- validate a SOZip file
- convert an existing Zip file into a SOZip optimized one

Synopsis

```
Usage: gdal vsi sozip <SUBCOMMAND> [OPTIONS]
where <SUBCOMMAND> is one of:
  - create:   Create a Seek-optimized ZIP (SOZIP) file.
  - list:     List content of a ZIP file, with SOZIP related information.
  - optimize: Create a Seek-optimized ZIP (SOZIP) file from a regular ZIP file.
  - validate: Validate a ZIP file, possibly using SOZIP optimization.

Common Options:
  -h, --help              Display help message and exit
  --json-usage            Display usage as JSON document and exit
  --config <KEY>=<VALUE>  Configuration option [may be repeated]
```

"gdal vsi sozip create"

Description

Adds one or several files to a new or existing zip file.

Synopsis

```
Usage: gdal vsi sozip create [OPTIONS] <INPUT> <OUTPUT>

Create a Seek-optimized ZIP (SOZIP) file.

Positional arguments:
  -i, --input <INPUT>                                        Input filenames [may be␣
↪repeated] [required]
  -o, --output <OUTPUT>                                      Output ZIP filename␣
↪[required]

Common Options:
  -h, --help                                                 Display help message and␣
↪exit
  --json-usage                                               Display usage as JSON␣
↪document and exit
  --config <KEY>=<VALUE>                                     Configuration option [may␣
↪be repeated]
  -q, --quiet                                                Quiet mode (no progress␣
↪bar or warning message) [not available in pipelines]

Options:
  --overwrite                                                Whether overwriting␣
↪existing output dataset is allowed
  -r, --recursive                                            Travels the directory␣
↪structure of the specified directories recursively
  -j, --junk-paths, --no-paths                               Store just the name of a␣
↪saved file, and do not store directory names
  --enable-sozip <ENABLE-SOZIP>                              Whether to automatically/
↪systematically/never apply the SOZIP optimization. ENABLE-SOZIP=auto|yes|no (default:␣
↪auto)
  --sozip-chunk-size <value in bytes or with K/M suffix>       Chunk size for a seek-
↪optimized file (default: 32768)
  --sozip-min-file-size <value in bytes or with K/M/G suffix>  Minimum file size to␣
↪decide if a file should be seek-optimized (default: 1 MB)
  --content-type <CONTENT-TYPE>                              Store the Content-Type of␣
↪the file being added.
```

Options

-i, --input <INPUT>

Input filenames. Required. Several file names or directory names accepted.

-o, --output <OUTPUT>

Output ZIP filename. Required. Must have a `.zip` extension

--overwrite

Whether overwriting existing output is allowed.

-r, --recursive

Travels the directory structure of the specified directories recursively.

-j, --junk-paths, --no-paths

Store just the name of a saved file (junk the path), and do not store directory names. By default, sozip will store the full path (relative to the current directory).

--enable-sozip auto|yes|no

In auto mode, a file is seek-optimized only if its size is above the value of *--sozip-min-file-size*. In yes mode, all input files will be seek-optimized. In no mode, no input files will be seek-optimized.

--sozip-chunk-size <value>

Chunk size for a seek-optimized file. Defaults to 32768 bytes. The value is specified in bytes, or K and M suffix (optionally preceded by a space) can be respectively used to specify a value in kilo-bytes or mega-bytes.

--sozip-min-file-size <value>

Minimum file size to decide if a file should be seek-optimized, in --enable-sozip=auto mode. Defaults to 1 MB byte. The value is specified in bytes, or K, M or G suffix (optionally preceded by a space) can be respectively used to specify a value in kilo-bytes, mega-bytes or giga-bytes.

--content-type <value>

Store the Content-Type for the file being added as a key-value pair in the extra field extension 'KV' (0x564b) dedicated to storing key-value pair metadata

-q, --quiet

Do not output any informative message (only errors).

Multithreading

The *GDAL_NUM_THREADS* configuration option can be set to ALL_CPUS or a integer value to specify the number of threads to use for SOZip-compressed files. Defaults to ALL_CPUS.

Examples

Example 1: Create a, potentially seek-optimized, ZIP file with the content of my.gpkg

```
gdal vsi sozip create my.gpkg my.gpkg.zip
```

Example 2: Create a, potentially seek-optimized, ZIP file from the content of a source directory:

```
gdal vsi sozip create -r source_dir/ my.gpkg.zip
```

"gdal vsi sozip optimize"

Description

Create a new zip file from the content of an existing one, possibly applying SOZip optimization when relevant.

Synopsis

```
Usage: gdal vsi sozip optimize [OPTIONS] <INPUT> <OUTPUT>

Create a Seek-optimized ZIP (SOZIP) file from a regular ZIP file.

Positional arguments:
  -i, --input <INPUT>                                        Input ZIP filename␣
```

(continues on next page)

(continued from previous page)

```
↪[required]
  -o, --output <OUTPUT>                                        Output ZIP filename␣
↪[required]

Common Options:
  -h, --help                                                   Display help message and␣
↪exit
  --json-usage                                                 Display usage as JSON␣
↪document and exit
  --config <KEY>=<VALUE>                                       Configuration option [may␣
↪be repeated]
  -q, --quiet                                                  Quiet mode (no progress␣
↪bar or warning message) [not available in pipelines]

Options:
  --overwrite                                                  Whether overwriting␣
↪existing output dataset is allowed
  --enable-sozip <ENABLE-SOZIP>                                Whether to automatically/
↪systematically/never apply the SOZIP optimization. ENABLE-SOZIP=auto|yes|no (default:␣
↪auto)
  --sozip-chunk-size <value in bytes or with K/M suffix>       Chunk size for a seek-
↪optimized file (default: 32768)
  --sozip-min-file-size <value in bytes or with K/M/G suffix>  Minimum file size to␣
↪decide if a file should be seek-optimized (default: 1 MB)
```

Options

-i, --input <INPUT>

Input ZIP filename. Required.

-o, --output <OUTPUT>

Output ZIP filename. Required. Must have a `.zip` extension

--overwrite

Whether overwriting existing output is allowed.

--enable-sozip `auto|yes|no`

In `auto` mode, a file is seek-optimized only if its size is above the value of *--sozip-chunk-size*. In `yes` mode, all input files will be seek-optimized. In `no` mode, no input files will be seek-optimized.

--sozip-chunk-size <value>

Chunk size for a seek-optimized file. Defaults to 32768 bytes. The value is specified in bytes, or K and M suffix can be respectively used to specify a value in kilo-bytes or mega-bytes.

--sozip-min-file-size <value>

Minimum file size to decide if a file should be seek-optimized, in --enable-sozip=auto mode. Defaults to 1 MB byte. The value is specified in bytes, or K, M or G suffix can be respectively used to specify a value in kilo-bytes, mega-bytes or giga-bytes.

-q, --quiet

Do not output any informative message (only errors).

Multithreading

The *GDAL_NUM_THREADS* configuration option can be set to ALL_CPUS or a integer value to specify the number of threads to use for SOZip-compressed files. Defaults to ALL_CPUS.

Examples

Example 3: Create a, potentially seek-optimized, ZIP file sozip_optimized.zip from an existing ZIP file in.zip.

```
gdal vsi sozip optimize in.zip sozip_optimized.zip
```

"gdal vsi sozip list"

Description

List the files contained in the zip file in an output similar to Info-ZIP **unzip** utility, but with the addition of a column indicating whether each file is seek-optimized.

Synopsis

```
Usage: gdal vsi sozip list [OPTIONS] <INPUT>

List content of a ZIP file, with SOZIP related information.

Positional arguments:
  -i, --input <INPUT>     Input ZIP filename [required]

Common Options:
  -h, --help              Display help message and exit
  --json-usage            Display usage as JSON document and exit
  --config <KEY>=<VALUE>  Configuration option [may be repeated]
```

Options

-i, --input <INPUT>

Input ZIP filename. Required.

Examples

Example 4: List contents of my.zip.

```
gdal vsi sozip list my.zip
```

"gdal vsi sozip validate"

Description

Validates a SOZip file. Baseline ZIP validation is done in a light way, limited to being able to browse through ZIP records with the InfoZIP-based ZIP reader used by GDAL. But validation of the SOZip-specific aspects is done more thoroughly.

Synopsis

```
Usage: gdal vsi sozip list [OPTIONS] <INPUT>

List content of a ZIP file, with SOZIP related information.

Positional arguments:
  -i, --input <INPUT>     Input ZIP filename [required]

Common Options:
  -h, --help              Display help message and exit
  --json-usage            Display usage as JSON document and exit
  --config <KEY>=<VALUE>  Configuration option [may be repeated]
```

Options

-i, --input <INPUT>

Input ZIP filename. Required.

-q, --quiet

Do not output any informative message (only errors).

-v, --verbose

Turn on verbose mode.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 5: Validate my.zip.

```
gdal vsi sozip validate my.zip
```

5.1.8 Driver specific commands

5.1.8.1 gdal driver cog validate

Added in version 3.13.

Synopsis

```
Usage: gdal driver cog validate [OPTIONS] <DATASET>

Validate if a TIFF file is a Cloud Optimized GeoTIFF

Positional arguments:
  --input, --dataset <DATASET>  COG dataset [required]

Common Options:
```

(continues on next page)

(continued from previous page)

```
  -h, --help                    Display help message and exit
  --json-usage                  Display usage as JSON document and exit
  --config <KEY>=<VALUE>        Configuration option [may be repeated]
  -q, --quiet                   Quiet mode (no progress bar or warning message) [not␣
↪available in pipelines]

Options:
  --full-check <FULL-CHECK>     Whether to perform full check. FULL-CHECK=auto|yes|no␣
↪(default: auto)
```

Description

Validate if a TIFF file is a *Cloud Optimized GeoTIFF* and emits a report.

> **Note**
>
> This program requires the GDAL Python bindings to be available.

Program-Specific Options

--full-check `auto|yes|no`

Whether extensive checks, verifying leading and trailer bytes of strips or tiles, must be performed. In `auto` mode (the default), they are performed for local file, but not for remote files

Standard Options

-q, --quiet

Suppress progress bar and some warning messages.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1

Check if a remote file is a valid COG file.

```
gdal driver cog validate /vsicurl/https://example.com/some.tif
```

5.1.8.2 gdal driver gpkg repack

Added in version 3.11.

Synopsis

```
Usage: gdal driver gpkg repack [OPTIONS] <DATASET>

Repack/vacuum in-place a GeoPackage dataset

Positional arguments:
  --dataset <DATASET>     GeoPackage dataset [required]

Common Options:
  -h, --help              Display help message and exit
  --json-usage            Display usage as JSON document and exit
  --config <KEY>=<VALUE>  Configuration option [may be repeated]
```

Description

Runs the `VACUUM` SQLite3 operation on a GeoPackage dataset to recover lost space due to updates or deletions.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1

```
gdal driver gpkg repack my.gpkg
```

5.1.8.3 gdal driver gpkg validate

Added in version 3.13.

Synopsis

```
Usage: gdal driver gpkg validate [OPTIONS] <DATASET>

Validate conformance of a GeoPackage dataset against the GeoPackage specification

Positional arguments:
  --dataset <DATASET>     GeoPackage dataset [required]

Common Options:
  -h, --help              Display help message and exit
  --json-usage            Display usage as JSON document and exit
  --config <KEY>=<VALUE>  Configuration option [may be repeated]
  -q, --quiet             Quiet mode (no progress bar or warning message) [not available␣
↪in pipelines]

Options:
  --full-check            Whether to perform full check
  -v, --verbose           Turn on verbose mode
```

Description

Validate if GeoPackage dataset conforms to the GeoPackage specification.

> **Note**
>
> This program requires the GDAL Python bindings to be available.

Program-Specific Options

--full-check

Enable extensive checks, that may go beyond what is mandated by the GeoPackage specification

Standard Options

-v, --verbose

Enable verbose mode.

-q, --quiet

Suppress progress bar and some warning messages.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1

Check if a GeoPackage file is conformant

```
gdal driver gpkg validate my.gpkg
```

5.1.8.4 gdal driver gti create

Added in version 3.11.

Synopsis

```
Usage: gdal driver gti create [OPTIONS] <INPUT>... <OUTPUT>

Create an index of raster datasets compatible with the GDAL Tile Index (GTI) driver.

Positional arguments:
  -i, --input <INPUT>                                          Input raster datasets␣
↪[may be repeated] [required]
  -o, --output <OUTPUT>                                        Output vector dataset␣
↪[required]

Common Options:
  -h, --help                                                   Display help message␣
```

(continues on next page)

(continued from previous page)

```
↪and exit
  --json-usage                                                  Display usage as JSON␣
↪document and exit
  --config <KEY>=<VALUE>                                        Configuration option␣
↪[may be repeated]
  -q, --quiet                                                   Quiet mode (no␣
↪progress bar or warning message) [not available in pipelines]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>           Output format
  --co, --creation-option <KEY>=<VALUE>                         Creation option [may␣
↪be repeated]
  --lco, --layer-creation-option <KEY>=<VALUE>                  Layer creation option␣
↪[may be repeated]
  --overwrite                                                   Whether overwriting␣
↪existing output dataset is allowed
                                                                Mutually exclusive␣
↪with --update
  --update                                                      Whether to open␣
↪existing dataset in update mode
                                                                Mutually exclusive␣
↪with --overwrite
  --overwrite-layer                                             Whether overwriting␣
↪existing output layer is allowed
  --append                                                      Whether appending to␣
↪existing layer is allowed
  --nln, --output-layer <OUTPUT-LAYER>                          Output layer name
  --recursive                                                   Whether input␣
↪directories should be explored recursively.
  --filename-filter <FILENAME-FILTER>                           Pattern that the␣
↪filenames in input directories should follow ('*' and '?' wildcard) [may be repeated]
  --min-pixel-size <MIN-PIXEL-SIZE>                             Minimum pixel size in␣
↪term of geospatial extent per pixel (resolution) that a raster should have to be␣
↪selected.
  --max-pixel-size <MAX-PIXEL-SIZE>                             Maximum pixel size in␣
↪term of geospatial extent per pixel (resolution) that a raster should have to be␣
↪selected.
  --location-name <LOCATION-NAME>                               Name of the field␣
↪with the raster path (default: location)
  --absolute-path                                               Whether the path to␣
↪the input datasets should be stored as an absolute path
  --output-crs <OUTPUT-CRS>                                     Output CRS
  --metadata <KEY>=<VALUE>                                      Add dataset metadata␣
↪item [may be repeated]
  --skip-errors                                                 Skip errors related␣
↪to input datasets
  --profile <PROFILE>                                           Profile of output␣
↪dataset. PROFILE=none|STAC-GeoParquet (default: none)
  --base-url <BASE-URL>                                         Base URL for STAC-
↪GeoParquet href
  --id-method <ID-METHOD>                                       How to derive STAC-
↪GeoParquet 'id'. ID-METHOD=filename|md5|metadata-item (default: filename)
```

(continues on next page)

(continued from previous page)

```
  --id-metadata-item <ID-METADATA-ITEM>                          Name of metadata item␣
↪used to set STAC-GeoParquet 'id' (default: id)
  --xml-filename <XML-FILENAME>                                  Filename of the XML␣
↪Virtual Tile Index file to generate, that can be used as an input for the GDAL GTI /␣
↪Virtual Raster Tile Index driver
  --resolution <xres>,<yres>                                     Resolution (in␣
↪destination CRS units) of the virtual mosaic
  --bbox <BBOX>                                                  Bounding box (in␣
↪destination CRS units) of the virtual mosaic
  --ot, --datatype, --output-data-type <OUTPUT-DATA-TYPE>        Datatype of the␣
↪virtual mosaic. OUTPUT-DATA-
↪TYPE=UInt8|Int8|UInt16|Int16|UInt32|Int32|UInt64|Int64|CInt16|CInt32|Float16|Float32|Float64|CFloat32
  --band-count <BAND-COUNT>                                      Number of bands of␣
↪the virtual mosaic
  --nodata <NODATA>                                              Nodata value(s) of␣
↪the bands of the virtual mosaic [may be repeated]
  --color-interpretation <COLOR-INTERPRETATION>                  Color␣
↪interpretation(s) of the bands of the virtual mosaic. COLOR-
↪INTERPRETATION=red|green|blue|alpha|gray|undefined [may be repeated]
  --mask                                                         Defines that the␣
↪virtual mosaic has a mask band
  --fetch-metadata <gdal-metadata-name>,<field-name>,<field-type>  Fetch a metadata item␣
↪from source rasters and write it as a field in the index. [may be repeated]
```

Description

gdal driver gti create creates a vector dataset with a record for each input raster file, an attribute containing the filename, and a polygon geometry outlining the raster, to be used as input for the *GTI* driver.

This is an extension of *gdal raster index*.

There are two possibilities:

- either use directly a vector tile index generated by **gdal driver gti create** as the input of the GTI driver
- or generate a small XML .gti wrapper file, for easier use with non-file-based formats such as databases, or for vector formats that do not support setting layer metadata items.

Formats that support layer metadata are for example GeoPackage (`--of GPKG`), FlatGeoBuf (`--of FlatGeoBuf`) or PostGIS (`--of PG`)

Setting *--resolution* and *--ot* is recommended to avoid the GTI driver to have to deduce them by opening the first tile in the index. If the tiles have nodata or mask band, *--nodata* and *--mask* should also be set.

In a GTI context, the extent of all tiles referenced in the tile index must be expressed in a single CRS. Consequently, if input tiles may have different CRS, *--output-crs* must be specified.

The following options are available:

Standard options

-f, --of, --format, --output-format <OUTPUT-FORMAT>

Which output vector format to use. Allowed values may be given by `gdal --formats | grep vector | grep rw | sort`

--co, --creation-option <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

May be repeated.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--formats* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

-lco <NAME>=<VALUE>

Layer creation option (format specific)

--overwrite

Allow program to overwrite existing target file or dataset. Otherwise, by default, **gdal** errors out if the target file or dataset already exists.

--update

Whether the output dataset must be opened in update mode. Implies that it already exists.

--overwrite-layer

Whether overwriting the existing output vector layer is allowed.

--append

Whether appending features to the existing output vector layer is allowed

-l, --nln, --layer <LAYER>

Provides a name for the output vector layer.

--recursive

Whether input directories should be explored recursively.

--filename-filter <FILENAME-FILTER>

Pattern that the filenames contained in input directories should follow. '*' is a wildcard character that matches any number of any characters including none. '?' is a wildcard character that matches a single character. Comparisons are done in a case insensitive way. Several filters may be specified.

For example: `--filename-filter "*.tif,*.tiff"`

--min-pixel-size <val>

Minimum pixel size in term of geospatial extent per pixel (resolution) that a raster should have to be selected. The pixel size is evaluated after reprojection of its extent to the target CRS defined by `--output-crs`.

--max-pixel-size <val>

Maximum pixel size in term of geospatial extent per pixel (resolution) that a raster should have to be selected. The pixel size is evaluated after reprojection of its extent to the target CRS defined by `--output-crs`.

-location-name <LOCATION-NAME>

The output field name to hold the file path/location to the indexed rasters. The default field name is `location`.

--absolute-path

The absolute path to the raster files is stored in the index file. By default the raster filenames will be put in the file exactly as they are specified on the command line.

--output-crs <OUTPUT-CRS>

Geometries of input files will be transformed to the desired target coordinate reference system. Default creates simple rectangular polygons in the same coordinate reference system as the input rasters.

--metadata <KEY>=<VALUE>

Write an arbitrary layer metadata item, for formats that support layer metadata. This option may be repeated.

--xml-filename <name>

Filename of the XML Virtual Tile Index file to generate, that can be used as an input for the GDAL GTI / Virtual Raster Tile Index driver. This can be useful when writing the tile index in a vector format that does not support writing layer metadata items.

--resolution <xres,<yres>

Target resolution in CRS unit per pixel.

Written in the XML Virtual Tile Index if *--xml-filename* is specified, or as RESX and RESY layer metadata items for formats that support layer metadata.

--bbox <xmin>,<ymin>,<xmax>,<ymax>

Target extent in CRS unit.

'x' is longitude values for geographic CRS and easting for projected CRS. 'y' is latitude values for geographic CRS and northing for projected CRS.

Written in the XML Virtual Tile Index if *--xml-filename* is specified, or as MINX, MINY, MAXX and MAXY layer metadata items for formats that support layer metadata.

--ot, --datatype, --output-data-type <datatype>

Data type of the tiles of the tile index: Byte, Int8, UInt16, Int16, UInt32, Int32, UInt64, Int64, Float32, Float64, CInt16, CInt32, CFloat32 or CFloat64

Written in the XML Virtual Tile Index if *--xml-filename* is specified, or as DATA_TYPE layer metadata item for formats that support layer metadata.

--band-count <val>

Number of bands of the tiles of the tile index.

Written in the XML Virtual Tile Index if *--xml-filename* is specified, or as BAND_COUNT layer metadata item for formats that support layer metadata.

A mix of tiles with N and N+1 bands is allowed, provided that the color interpretation of the (N+1)th band is alpha. The N+1 value must be written as the band count in that situation.

If *--nodata* or *--color-interpretation* are specified and have multiple values, the band count is also inferred from that number.

--nodata <val>[,<val>...]

Nodata value of the tiles of the tile index.

Written in the XML Virtual Tile Index if *--xml-filename* is specified, or as NODATA layer metadata item for formats that support layer metadata.

--color-interpretation <val>[,<val>...]

Color interpretation of of the tiles of the tile index: red, green, blue, alpha, gray, undefined.

Written in the XML Virtual Tile Index if *--xml-filename* is specified, or as COLOR_INTERPRETATION layer metadata item for formats that support layer metadata.

--mask

Whether tiles in the tile index have a mask band.

Written in the XML Virtual Tile Index if *--xml-filename* is specified, or as MASK_BAND layer metadata item for formats that support layer metadata.

--fetched-metadata <gdal_md_name>,<fld_name>,<fld_type>

Fetch a metadata item from the raster tile and write it as a field in the tile index.

<gdal_md_name> should be the name of the raster metadata item. {PIXEL_SIZE} may be used as a special name to indicate the pixel size.

<fld_name> should be the name of the field to create in the tile index.

<fld_type> should be the name of the type to create. One of String, Integer, Integer64, Real, Date, DateTime

This option may be repeated.

For example: --fetched-metadata TIFFTAG_DATETIME,creation_date,DateTime

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1

Make a tile index from GeoTIFF files, with metadata suitable for use by the GDAL GTI / Virtual Raster Tile Index driver.

```
gdal driver gti create --ot Byte --resolution=60,60 --band-count=3 --color-
↪interpretation=Red,Green,Blue *.tif tile_index.gti.gpkg
```

5.1.8.5 gdal driver openfilegdb repack

Added in version 3.11.

Synopsis

```
Usage: gdal driver openfilegdb repack [OPTIONS] <DATASET>

Repack a FileGeoDatabase dataset

Positional arguments:
  --dataset <DATASET>     FileGeoDatabase dataset [required]

Common Options:
  -h, --help              Display help message and exit
  --json-usage            Display usage as JSON document and exit
  --config <KEY>=<VALUE>  Configuration option [may be repeated]
  -q, --quiet             Quiet mode (no progress bar or warning message) [not available␣
↪in pipelines]
```

Description

Runs the a repack operation on a FileGeodatabase dataset to recover lost space due to updates or deletions.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1

```
gdal driver openfilegdb repack my.gdb
```

5.1.8.6 gdal driver parquet create-metadata-file

Added in version 3.13.

Synopsis

```
Usage: gdal driver parquet create-metadata-file [OPTIONS] <INPUT>... <OUTPUT>

Create the _metadata file for a partitioned dataset

Positional arguments:
  --input <INPUT>         Input Parquet datasets (created by algorithm) [1.. values]␣
↪[required]
  --output <OUTPUT>       Output Parquet dataset [required]

Common Options:
  -h, --help              Display help message and exit
  --json-usage            Display usage as JSON document and exit
  --config <KEY>=<VALUE>  Configuration option [may be repeated]

Options:
  --overwrite             Whether overwriting existing output is allowed
```

Description

Creates a `_metadata` Parquet metadata-only file that contains the schema of the partitioned Parquet files and the path to them.

This program is automatically called by *gdal vector partition* for a Parquet output.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1

```
gdal driver parquet create-metadata-file --input partitioned/0.parquet \
            --input partitioned/1.parquet --output partitioned/_metadata
```

5.1.8.7 gdal driver pdf list-layer

Added in version 3.11.

Synopsis

```
Usage: gdal driver pdf list-layers [OPTIONS] <INPUT>

List layers of a PDF dataset

Positional arguments:
  -i, --input <INPUT>                                    Input raster or vector dataset␣
↪[required]

Common Options:
  -h, --help                                             Display help message and exit
  --json-usage                                           Display usage as JSON document␣
↪and exit
  --config <KEY>=<VALUE>                                 Configuration option [may be␣
↪repeated]

Options:
  -f, --of, --format, --output-format <OUTPUT-FORMAT>  Output format. OUTPUT-
↪FORMAT=json|text (default: json)
```

Description

Return the list of layers of a PDF file, as a JSON array or a text list.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Return the list of layers of a PDF file as a JSON array

```
gdal driver pdf list-layers autotest/gdrivers/data/pdf/adobe_style_geospatial.pdf
```

returns:

```
[
  "New_Data_Frame",
  "New_Data_Frame.Graticule",
  "Layers",
  "Layers.Measured_Grid",
  "Layers.Graticule"
]
```

Example 2: Return the list of layers of a PDF file as as a text list

```
gdal driver pdf list-layers --of=text autotest/gdrivers/data/pdf/adobe_style_geospatial.
↪pdf
```

returns:

```
New_Data_Frame
New_Data_Frame.Graticule
Layers
Layers.Measured_Grid
Layers.Graticule
```

5.1.8.8 gdal driver zarr add-georeferencing-convention

Added in version 3.14.

Synopsis

```
Usage: gdal driver zarr add-georeferencing-convention [OPTIONS] <INPUT> <CONVENTION>

Add a georeferencing convention to an existing ZARR dataset

Positional arguments:
  -i, --input <INPUT>        Input multidimensional raster dataset [required]
  --convention <CONVENTION>  Georeferencing convention. CONVENTION=GDAL|spatial_proj␣
↪[required]

Common Options:
  -h, --help                 Display help message and exit
  --json-usage               Display usage as JSON document and exit
  --config <KEY>=<VALUE>     Configuration option [may be repeated]
```

Description

Add (but not replace) a georeferencing convention to an existing ZARR dataset.

This has only effect on arrays for which georeferencing is recognized.

This can been used for example to add the Zarr spatial and geo-proj conventions by specifying `--convention` to `spatial_proj`.

Program-Specific Options

--convention `GDAL|spatial_proj`

Name of the georeferencing convention to add.

`GDAL` uses a `_CRS` attribute (cf *GDAL convention*).

`spatial_proj` uses the Zarr spatial and geo-proj conventions.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1

title
Add the `spatial_proj` convention.

```
gdal driver zarr add-georeferencing-conventions my.zarr spatial_proj
```

5.2 "Traditional" applications

5.2.1 General

5.2.1.1 Syntax of arguments of command-line utilities

The following table describes the notation used to document the syntax of arguments of command-line utilities.

Notation	Description
Text without brackets or braces	Text that must be typed as shown
`<value>`	Placeholder for which a value must be substituted.
[Text inside square brackets]	Optional item.
{ Text inside braces }	Set of required items, separated by a vertical bar.
Vertical bar (`|`)	Separator between mutually exclusive items. At least one must be specified.
Ellipsis (`...`)	Placed after an item that can be repeated.

Example

Given the following synopsis:

```
my_utility [-wkt_format {WKT1|WKT2}] [-oo <NAME>=<VALUE>]...
           <datasetname>
```

The following command lines are valid:

```
my_utility foo.tif
```

```
my_utility -wkt_format WKT1 foo.tif
```

```
my_utility -oo FOO=BAR -oo BAR=BAZ foo.tif
```

5.2.2 Raster programs

5.2.2.1 Common options for raster programs

All GDAL command line programs support the following common options.

--version

Report the version of GDAL and exit.

--build

Report detailed information about GDAL in use.

--formats

List all raster formats supported by this GDAL build (read-only and read-write) and exit.

The format support is indicated as follows:

- `ro` is read-only driver
- `rw` is read or write (i.e. supports *GDALDriver::CreateCopy()*)
- `rw+` is read, write and update (i.e. supports *GDALDriver::Create()*)
- A `v` is appended for formats supporting virtual IO (/vsimem, /vsigzip, /vsizip, etc).
- A `s` is appended for formats supporting subdatasets.

The order in which drivers are listed is the one in which they are registered, which determines the order in which they are successively probed when opening a dataset. Most of the time, this order does not matter, but in some situations, several drivers may recognize the same file. The `-if` option of some utilities can be specified to restrict opening the dataset with a subset of drivers (generally one). Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions. Alternatively, the *GDAL_SKIP* configuration option can also be used to exclude one or several drivers.

--format `<format>`

List detailed information about a single format driver. The format should be the short name reported in the --formats list, such as GTiff.

--formats

List all drivers. Can be combined with `-json` switch of **gdalinfo** of since GDAL 3.10

--optfile `<filename>`

Read the named file and substitute the contents into the command line options list. Lines beginning with # will be ignored. Multi-word arguments may be kept together with double quotes. Multiple --optfiles may be used: --optfile file1 --optfile file2...

--config `<key> <value>`

Sets the named configuration keyword to the given value, as opposed to setting them as environment variables. Some common configuration keywords are *GDAL_CACHEMAX* (memory used internally for caching in megabytes) and *GDAL_DATA* (path of the GDAL "data" directory). Individual drivers may be influenced by other *configuration options*.

--debug `<value>`

Control what debugging messages are emitted. A value of ON will enable all debug messages. A value of OFF will disable all debug messages. Another value will select only debug messages containing that string in the debug prefix code.

--help-general

Gives a brief usage message for the generic GDAL command line options and exit.

Creating new files

Access an existing file to read it is generally quite simple. Just indicate the name of the file or dataset on the command line. However, creating a file is more complicated. It may be necessary to indicate the format to create, various creation options affecting how it will be created and perhaps a coordinate system to be assigned. Many of these options are handled similarly by different GDAL utilities, and are introduced here.

-of <format>

Select the format to create the new file as. The formats are assigned short names such as GTiff (for GeoTIFF) or HFA (for Erdas Imagine). The list of all format codes can be listed with the *--formats* switch. Only formats list as (rw) (read-write) can be written.

If not specified, the format is guessed from the extension.

-co <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--format* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

-a_srs <srs>

-s_srs <srs>

-t_srs <srs>

Several utilities (e.g. **gdal_translate** and **gdalwarp**) include the ability to specify coordinate systems with command line options like *-a_srs* (assign SRS to output), *-s_srs* (source SRS) and *-t_srs* (target SRS). These utilities allow the coordinate system (SRS = spatial reference system) to be assigned in a variety of formats.

- NAD27|NAD83|WGS84|WGS72

  These common geographic (lat/long) coordinate systems can be used directly by these names.

- EPSG:n

  Coordinate systems (projected or geographic) can be selected based on their EPSG codes. For instance, EPSG:27700 is the British National Grid. A list of EPSG coordinate systems can be found in the GDAL data files gcs.csv and pcs.csv.

- PROJ.4 definition

  A PROJ.4 definition string can be used as a coordinate system. Take care to keep the proj.4 string together as a single argument to the command (usually by double quoting).

  For instance +proj=utm +zone=11 +datum=WGS84.

- OpenGIS Well Known Text

  The Open GIS Consortium has defined a textual format for describing coordinate systems as part of the Simple Features specifications. This format is the internal working format for coordinate systems used in GDAL. The name of a file containing a WKT coordinate system definition may be used a coordinate system argument, or the entire coordinate system itself may be used as a command line option (though escaping all the quotes in WKT is quite challenging).

- ESRI Well Known Text

  ESRI uses a slight variation on OGC WKT format in their ArcGIS product (ArcGIS .prj files), and these may be used in a similar manner o WKT files, but the filename should be prefixed with ESRI::.

  For example, *"ESRI::NAD 1927 StatePlane Wyoming West FIPS 4904.prj"*.

- Spatial References from URLs

  For example http://spatialreference.org/ref/user/north-pacific-albers-conic-equal-area/.

- `filename`

  File containing WKT, PROJ.4 strings, or XML/GML coordinate system definitions can be provided.

5.2.2.2 gdal-config (Unix)

Determines various information about a GDAL installation.

Synopsis

```
gdal-config [OPTIONS]
Options:
        [--prefix[=DIR]]
        [--libs]
        [--cflags]
        [--version]
        [--ogr-enabled]
        [--formats]
```

Description

This utility script (available on Unix systems) can be used to determine various information about a GDAL installation. It is normally just used by configure scripts for applications using GDAL but can be queried by an end user.

--prefix

the top level directory for the GDAL installation.

--libs

The libraries and link directives required to use GDAL.

--cflags

The include and macro definition required to compiled modules using GDAL.

--version

Reports the GDAL version.

--ogr-enabled

Reports "yes" or "no" to standard output depending on whether OGR is built into GDAL.

--formats

Reports which formats are configured into GDAL to stdout.

5.2.2.3 gdal2tiles

Synopsis

```
gdal2tiles [--help] [--help-general]
              [--legacy]
              [-p <profile>] [-r resampling] [-s <srs>] [-z <zoom>]
              [-e] [-a nodata] [-v] [-q] [-h] [-k] [-n] [-u <url>]
              [-w <webviewer>] [-t <title>] [-c <copyright>]
              [--processes=<NB_PROCESSES>] [--mpi] [--xyz]
              [--tilesize=<PIXELS>] --tiledriver=<DRIVER> [--tmscompatible]
              [--excluded-values=<EXCLUDED_VALUES>]
```

(continues on next page)

(continued from previous page)

```
                [--excluded-values-pct-threshold=<EXCLUDED_VALUES_PCT_THRESHOLD>]
                [--nodata-values-pct-threshold=<NODATA_VALUES_PCT_THRESHOLD>]
                [-g <googlekey] [-b <bingkey>] <input_file> [<output_dir>] [<COMMON_
↪OPTIONS>]
```

Description

Warning

Starting with GDAL 3.13, **gdal2tiles** is deprecated, and by default is remapped to *gdal raster tile*. For now, you may force the use of the legacy code by specifying *--legacy*, but legacy mode will be removed in GDAL 3.15. If you find yourself to need legacy mode and cannot find workarounds using gdal raster tile, please file a ticket at https://github.com/OSGeo/GDAL

This utility generates a directory with small tiles and metadata, following the OSGeo Tile Map Service Specification. Simple web pages with viewers based on Google Maps, OpenLayers and Leaflet are generated as well - so anybody can comfortably explore your maps on-line and you do not need to install or configure any special software (like MapServer) and the map displays very fast in the web browser. You only need to upload the generated directory onto a web server.

GDAL2Tiles also creates the necessary metadata for Google Earth (KML SuperOverlay), in case the supplied map uses EPSG:4326 projection.

World files and embedded georeferencing is used during tile generation, but you can publish a picture without proper georeferencing too.

Note

Inputs with non-Byte data type (i.e. `Int16`, `UInt16`,...) will be clamped to the `Byte` data type, causing wrong results. To avoid this it is necessary to rescale input to the `Byte` data type using *gdal_translate* utility.

Note

Config options of the input drivers may have an effect on the output of gdal2tiles. An example driver config option is GDAL_PDF_DPI, which can be found at *Configuration options*

Note

gdal2tiles is a Python utility, and is only available if GDAL Python bindings are available.

Tip

Equivalent in new "gdal" command line interface:

See *gdal raster tile*

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

--legacy

Added in version 3.13.

Force the use of the legacy code base. Since GDAL 3.13, by default, gdal2tiles is implemented using *gdal raster tile*.

-p <PROFILE>, **--profile**=<PROFILE>

Tile cutting profile (mercator, geodetic, raster) - default 'mercator' (Google Maps compatible).

Starting with GDAL 3.2, additional profiles are available from tms_XXXX.json files placed in GDAL data directory (provided all zoom levels use same origin, tile dimensions, and resolution between consecutive zoom levels vary by a factor of two).

-r <RESAMPLING>, **--resampling**=<RESAMPLING>

Resampling method (average, near, bilinear, cubic, cubicspline, lanczos, mode, max, min, med, q1, q3) - default 'average'.

-s <SRS>, **--s_srs**=<SRS>

The spatial reference system used for the source input data.

--xyz

Generate XYZ tiles (OSM Slippy Map standard) instead of TMS. In the default mode (TMS), tiles at y=0 are the southern-most tiles, whereas in XYZ mode (used by OGC WMTS too), tiles at y=0 are the northern-most tiles.

Added in version 3.1.

-d, --tmscompatible

When using the geodetic profile, specifies the base resolution as 0.703125 or 2 tiles at zoom level 0.

-z <ZOOM>, **--zoom**=<ZOOM>

Zoom levels to render (format:'2-5', '10-' or '10').

-e, --resume

Resume mode. Generate only missing files.

-a <NODATA>, **--srcnodata**=<NODATA>

Value in the input dataset considered as transparent. If the input dataset had already an associate nodata value, it is overridden by the specified value.

-v, --verbose

Generate verbose output of tile generation.

Starting with GDAL 3.7, that verbose output is emitted through the `logging.getLogger("gdal2tiles")` object.

-x, --exclude

Exclude transparent tiles from result tileset.

-q, --quiet

Disable messages and status to stdout

--processes=<NB_PROCESSES>

Number of parallel processes to use for tiling, to speed-up the computation.

--mpi

Added in version 3.5.

Assume launched by mpiexec, enable MPI parallelism and ignore --processes. Requires working MPI environment and the MPI for Python (mpi4py) package. User should set GDAL_CACHEMAX to an appropriate cache size per process based on memory per node and the number of processes launched per node.

> **Warning**
>
> --mpi mode is not supported in *gdal raster tile* non-legacy mode. You may specify `--legacy` to go on, but this legacy mode is going to be removed in GDAL 3.15.

--tilesize=<PIXELS>

Added in version 3.1.

Width and height in pixel of a tile. Default is 256.

--tiledriver=<DRIVER>

Which output driver to use for the tiles, determines the file format of the tiles. Currently PNG, WEBP and JPEG (JPEG added in GDAL 3.9) are supported. Default is PNG. Additional configuration for the WEBP and JPEG drivers are documented below.

Added in version 3.6.

--excluded-values=<EXCLUDED_VALUES>

Comma-separated tuple of values (thus typically "R,G,B"), that are ignored as contributing source pixels during resampling. The number of values in the tuple must be the same as the number of bands, excluding the alpha band. Several tuples of excluded values may be specified using the "(R1,G1,B2),(R2,G2,B2)" syntax. Only taken into account by Average currently. This concept is a bit similar to nodata/alpha, but the main difference is that pixels matching one of the excluded value tuples are still considered as valid, when determining the target pixel validity/density.

Added in version 3.9.

--excluded-values-pct-threshold=EXCLUDED_VALUES_PCT_THRESHOLD

Minimum percentage of source pixels that must be set at one of the --excluded-values to cause the excluded value, that is in majority among source pixels, to be used as the target pixel value. Default value is 50(%)

Added in version 3.9.

--nodata-values-pct-threshold=<NODATA_VALUES_PCT_THRESHOLD>

Minimum percentage of source pixels that must be at nodata (or alpha=0 or any other way to express transparent pixel) to cause the target pixel value to be transparent. Default value is 100 (%), which means that a target pixel is transparent only if all contributing source pixels are transparent. Only taken into account for average resampling.

Added in version 3.9.

-h, --help

Show help message and exit.

--version

Show program's version number and exit.

KML (Google Earth) options

Options for generated Google Earth SuperOverlay metadata

-k, --force-kml

Generate KML for Google Earth - default for 'geodetic' profile and 'raster' in EPSG:4326. For a dataset with different projection use with caution!

-n, --no-kml

Avoid automatic generation of KML files for EPSG:4326.

-u <URL>, **--url**=<URL>

URL address where the generated tiles are going to be published.

Web viewer options

Options for generated HTML viewers a la Google Maps

-w <WEBVIEWER>, **--webviewer**=<WEBVIEWER>

Web viewer to generate (all, google, openlayers, leaflet, mapml, none) - default 'all'.

-t <TITLE>, **--title**=<TITLE>

Title of the map.

-c <COPYRIGHT>, **--copyright**=<COPYRIGHT>

Copyright for the map.

-g <GOOGLEKEY>, **--googlekey**=<GOOGLEKEY>

Google Maps API key from https://developers.google.com/maps/get-started.

-b <BINGKEY>, **--bingkey**=<BINGKEY>

Bing Maps API key from https://www.bingmapsportal.com/

MapML options

MapML support is new to GDAL 3.2. When --webviewer=mapml is specified, --xyz is implied, as well as --tmscompatible if --profile=geodetic.

The following profiles are supported:

- mercator: mapped to OSMTILE MapML tiling scheme
- geodetic: mapped to WGS84 MapML tiling scheme
- APSTILE: from the tms_MapML_APSTILE.json data file

The generated MapML file in the output directory is `mapml.mapml`

Available options are:

--mapml-template=<filename>

Filename of a template mapml file where variables will be substituted. If not specified, the generic template_tiles.mapml file from GDAL data resources will be used

The --url option is also used to substitute `${URL}` in the template MapML file.

WEBP options

WEBP tiledriver support is new to GDAL 3.6. It is enabled by using --tiledriver=WEBP.

The following configuration options are available to further customize the WebP output:

--webp-quality=<QUALITY>
: QUALITY is a integer between 1-100. Default is 75.

--webp-lossless
: Use WEBP lossless compression, default is lossy

> **Note**
>
> GDAL *WEBP driver* documentation can be consulted

JPEG options

JPEG tiledriver support is new to GDAL 3.9. It is enabled by using --tiledriver=JPEG.

Note that JPEG does not support transparency, hence edge tiles will display black pixels in areas not covered by the source raster.

The following configuration options are available to further customize the JPEG output:

---jpeg-quality=JPEG_QUALITY
: QUALITY is a integer between 1-100. Default is 75.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Basic example

```
gdal2tiles --zoom=2-5 input.tif output_folder
```

Example 2: MapML generation

```
gdal2tiles --zoom=16-18 -w mapml -p APSTILE --url "https://example.com" input.tif output_
↪folder
```

Example 3: MPI example

```
mpiexec -n $NB_PROCESSES gdal2tiles --mpi --config GDAL_CACHEMAX 500 --zoom=2-
↪5 input.tif output_folder
```

5.2.2.4 gdal2xyz

Synopsis

```
gdal2xyz [--help] [--help-general]
    [-skip <factor>]
    [-srcwin <xoff> <yoff> <xsize> <ysize>]
    [-b <band>]... [-allbands]
    [-skipnodata]
    [-csv]
    [-srcnodata <value>] [-dstnodata <value>]
    <src_dataset> <dst_dataset>
```

Description

The **gdal2xyz** utility can be used to translate a raster file into xyz format. *gdal2xyz* can be used as an alternative to *gdal_translate of=xyz*. Features include:

- Select more then one band
- Skip or replace nodata value
- Return the output as numpy arrays.

Note

gdal2xyz is a Python utility, and is only available if GDAL Python bindings are available.

Tip

Equivalent in new "gdal" command line interface:

See *gdal raster convert* with `--of XYZ`.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-skip

How many rows/cols to skip in each iteration.

-srcwin <xoff> <yoff> <xsize> <ysize>

Selects a subwindow from the source image for copying based on pixel/line location.

-b, -band <band>

Select band *band* from the input spectral bands for output. Bands are numbered from 1 in the order spectral bands are specified. Multiple **-b** switches may be used. When no -b switch is used, the first band will be used. In order to use all input bands set *-allbands* or *-b 0*.

-allbands

Select all input bands.

-csv

Use comma instead of space as a delimiter.

-skipnodata

Exclude the output lines with nodata value (as determined by srcnodata)

-srcnodata

The nodata value of the dataset (for skipping or replacing) Default (*None*) - Use the dataset nodata value; *Sequence/Number* - Use the given nodata value (per band or per dataset).

-dstnodata

Replace source nodata with a given nodata. Has an effect only if not setting *-skipnodata*. Default(*None*) - Use *srcnodata*, no replacement; *Sequence/Number* - Replace the *srcnodata* with the given nodata value (per band or per dataset).

-h, --help

Show help message and exit.

<src_dataset>

The source dataset name. It can be either file name, URL of data source or subdataset name for multi-dataset files.

<dst_dataset>

The destination file name.

Examples

Example 1

```
gdal2xyz -b 1 -b 2 -dstnodata 0 input.tif output.txt
```

To create a text file in *xyz* format from the input file *input.tif*. The first columns, x and y, are the coordinates of the centers of each cell. The remaining columns represent the first and second bands. We also replace the dataset nodata values with zeros.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Caveats

gdal2xyz output values with a limited precision. Use `gdal_translate -of XYZ` if more precision is desired.

5.2.2.5 gdal_calc

Synopsis

```
gdal_calc [--help] [--help-general]
             --calc=expression --outfile=<out_filename> [-A <filename>]
             [--A_band=<n>] [-B...-Z <filename>] [<other_options>]
```

DESCRIPTION

Command line raster calculator with numpy syntax. Use any basic arithmetic supported by numpy arrays such as +, -, *, and / along with logical operators such as >. Note that all files must have the same dimensions (unless the `--extent` option is used), but no projection checking is performed (unless the `--projectionCheck` option is used).

> **Note**
>
> gdal_calc is a Python utility, and is only available if GDAL Python bindings are available.

> **Tip**
>
> Equivalent in new "gdal" command line interface:
>
> See *gdal raster calc*.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

--calc=<expression>

Calculation in numpy syntax using +, -, /, *, or any numpy array functions (i.e. `log10()`). Multiple `--calc` options can be listed to produce a multiband file (GDAL >= 3.2). See *Example 10*.

-A <filename>

Input GDAL raster file, you can use any letter (a-z, A-Z). (lower case supported since GDAL 3.3)

A letter may be repeated, or several values (separated by space) can be provided (GDAL >= 3.3). The effect will be to create a 3D numpy array. Since GDAL 3.5, wildcard exceptions (using ?, *) are supported for all shells/platforms. In such a case, the calculation formula must use this input as a 3D array and must return a 2D array (see *Example 3*). If the calculation does not return a 2D array an error will be generated.

--A_band=<n>

Number of raster band for file A (default 1).

--outfile=<filename>

Output file to generate or fill.

--NoDataValue=<value>

Output NoDataValue (default datatype specific value). To indicate not setting a NoDataValue use --NoDataValue=none (GDAL >= 3.3)

> **Note**
>
> Using the Python API: `None` value will indicate default datatype specific value. `'none'` value will indicate not setting a NoDataValue.

--hideNoData

Added in version 3.3.

Ignores the input bands NoDataValue. By default, the input bands NoDataValue are not participating in the calculation. By setting this setting - no special treatment will be performed on the input NoDataValue. and they

will be participating in the calculation as any other value. The output will not have a set NoDataValue, unless you explicitly specified a specific value by setting --NoDataValue=<value>.

--type=<datatype>

Output datatype, must be one of [`Byte`, `Int8`, `UInt16`, `Int16`, `UInt32`, `Int32`, `UInt64`, `Int64`, `Float64`, `Float32`, `CInt16`, `CInt32`, `CFloat64`, `CFloat32`].

> **Note**
>
> Despite the datatype set using `--type`, when doing intermediate arithmetic operations using operands of the same type, the operation result will honor the original datatype. This may lead into unexpected results in the final result.

> **Note**
>
> UInt64, Int64, CInt16, CInt32, CFloat32, CFloat64 have been added in GDAL 3.5.3 Int8 has been added in GDAL 3.7

--format=<gdal_format>

GDAL format for output file.

--color-table=<filename>

Allows specifying a filename of a color table (or a ColorTable object) (with Palette Index interpretation) to be used for the output raster. Supported formats: txt (i.e. like gdaldem, but color names are not supported), qlr, qml (i.e. exported from QGIS)

--extent=<option>

Added in version 3.3.

This option determines how to handle rasters with different extents. This option is mutually exclusive with the *projwin* option, which is used for providing a custom extent.

For all the options below the pixel size (resolution) and SRS (Spatial Reference System) of all the input rasters must be the same.

`ignore` (default) - only the dimensions of the rasters are compared. if the dimensions do not agree the operation will fail.

`fail` - the dimensions and the extent (bounds) of the rasters must agree, otherwise the operation will fail.

`union` - the extent (bounds) of the output will be the minimal rectangle that contains all the input extents.

`intersect` - the extent (bounds) of the output will be the maximal rectangle that is contained in all the input extents.

--projwin <ulx> <uly> <lrx> <lry>

Added in version 3.3.

This option provides a custom extent for the output, it is mutually exclusive with the *extent* option.

--projectionCheck

Added in version 3.3.

By default, no projection checking will be performed. By setting this option, if the projection is not the same for all bands then the operation will fail.

--creation-option=<option>

Passes a creation option to the output format driver. Multiple options may be listed. See format specific documentation for legal creation options for each format.

--co=<option>

The same as *creation-option*.

--allBands=[a-z, **A-Z**]

Apply the expression to all bands of a given raster. When `--allBands` is used, `--calc` may be specified only once. See *Example 11* and *Example 12*.

--overwrite

Overwrite output file if it already exists. Overwriting must be understood here as deleting and recreating the file from scratch. Note that if this option is *not* specified and the output file already exists, it will be updated in place.

--debug

Print debugging information.

--quiet

Suppress progress messages.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Python options

Added in version 3.3.

The following options are available by using the Python interface of **gdal_calc**. They are not available using the command prompt.

user_namespace

A dictionary of custom functions or other names to be available for use in the Calc expression.

return_ds

If enabled, the output dataset will be returned from the function and not closed.

color_table

Allows specifying a ColorTable object (with Palette Index interpretation) to be used for the output raster.

Examples

Example 1: Average of two files

```
gdal_calc -A input1.tif -B input2.tif --outfile=result.tif --calc="(A+B)/2"
```

Caution

If A and B inputs both have integer data types, integer division will be performed. To avoid this, you can convert one of the operands to a floating point type before the division operation.

```
gdal_calc -A input.tif -B input2.tif --outfile=result.tif --calc="(A.astype(numpy.
↪float64) + B) / 2"
```

Example 2: Summing three files

```
gdal_calc -A input1.tif -B input2.tif -C input3.tif --outfile=result.tif --calc="A+B+C"
```

Example 3: Combining three files into a 3D array and summing

```
gdal_calc -A input1.tif -A input2.tif -A input3.tif --outfile=result.tif --calc="numpy.
↪sum(A,axis=0)".
```

Example 4: Average of three files

```
gdal_calc -A input1.tif -B input2.tif -C input3.tif --outfile=result.tif --calc="(A+B+C)/
↪3"
```

Example 5: Average of three files, using 3D array

```
gdal_calc -A input1.tif input2.tif input3.tif --outfile=result.tif --calc="numpy.
↪average(a,axis=0)".
```

Example 6: Maximum of three files

```
gdal_calc -A input1.tif -B input2.tif -C input3.tif --outfile=result.tif --calc="numpy.
↪max((A,B,C),axis=0)"
```

Example 7: Maximum of three files, using a 3D array

```
gdal_calc -A input1.tif input2.tif input3.tif --outfile=result.tif --calc="numpy.max(A,
↪axis=0)"
```

Example 8: Setting values of zero and below to NODATA

```
gdal_calc -A input.tif --outfile=result.tif --calc="A*(A>0)" --NoDataValue=0
```

Example 9: Using logical operator to keep a range of values from input

```
gdal_calc -A input.tif --outfile=result.tif --calc="A*logical_and(A>100,A<150)"
```

Example 10: Performing two calculations and storing results in separate bands

```
gdal_calc -A input.tif --A_band=1 -B input.tif --B_band=2 \
  --outfile=result.tif --calc="(A+B)/2" --calc="B*logical_and(A>100,A<150)"
```

Example 11: Add a raster to each band in a 3-band raster

```
gdal_calc -A 3band.tif -B 1band.tif --outfile result.tif --calc "A+B" --allBands A
```

The result will have three bands, where each band contains the values of `1band.tif` added to the corresponding band in `3band.tif`.

Example 12: Add two three-band rasters

```
gdal_calc -A 3band_a.tif -B 3band_b.tif --outfile result.tif --calc "A+B" --allBands A --
↪allBands B
```

The result will have three bands, where each band contains the values of the corresponding band of `3band_a.tif` added to the corresponding band of `3band_b.tif`.

5.2.2.6 gdal_contour

Synopsis

```
gdal_contour [--help] [--help-general]
             [-b <band>] [-a <attribute_name>] [-amin <attribute_name>] [-amax
↪<attribute_name>]
             [-3d] [-inodata] [-snodata <n>] [-f <formatname>] [-i <interval>]
             [-dsco <NAME>=<VALUE>]... [-lco <NAME>=<VALUE>]...
             [-off <offset>] [-fl <level> <level>...] [-e <exp_base>]
             [-nln <outlayername>] [-q] [-p] [-gt <n>|unlimited]
             <src_filename> <dst_filename>
```

Description

The **gdal_contour** generates a vector contour file from the input raster elevation model (DEM).

The contour line-strings are oriented consistently and the high side will be on the right, i.e. a line string goes clockwise around a top.

> **Tip**
>
> Equivalent in new "gdal" command line interface:
>
> See *gdal raster contour*.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-q, --quiet

Suppress progress messages and other non-error output.

-b <band>

Picks a particular band to get the DEM from. Defaults to band 1.

-a <name>

Provides a name for the attribute in which to put the elevation. If not provided no elevation attribute is attached. Ignored in polygonal contouring (*-p*) mode.

-amin <name>

Provides a name for the attribute in which to put the minimum elevation of contour polygon. If not provided no minimum elevation attribute is attached. Ignored in default line contouring mode.

-amax <name>

Provides a name for the attribute in which to put the maximum elevation of contour polygon. If not provided no maximum elevation attribute is attached. Ignored in default line contouring mode.

-3d

Force production of 3D vectors instead of 2D. Includes elevation at every vertex.

-inodata

Ignore any nodata value implied in the dataset - treat all values as valid. Cannot be used together with *-snodata*.

-snodata <value>

Input pixel value to treat as "nodata". Cannot be used together with *-inodata*.

-f <format>

Create output in a particular format.

If not specified, the format is guessed from the extension (previously was ESRI Shapefile).

-dsco <NAME>=<VALUE>

Dataset creation option (format specific)

-lco <NAME>=<VALUE>

Layer creation option (format specific)

-i <interval>

Elevation interval between contours. Must specify either *-i* or *-fl* or *-e*. The first contour will be generated at the first multiple of `interval` which is greater than the raster minimum value.

-off <offset>

Offset from zero relative to which to interpret intervals.

For example, *-i 100* requests contours at ...-100, 0, 100... Further adding *-off 25* makes that request instead ...-75, 25, 125...

-fl <level>

Name one or more "fixed levels" to extract, in ascending order separated by spaces.

-e <base>

Generate levels on an exponential scale: *base* ^ *k*, for *k* an integer. Must specify either -i or -fl or -e.

-nln <name>

Provide a name for the output vector layer. Defaults to "contour".

-p

Generate contour polygons rather than contour lines.

When this mode is selected the polygons are created for values between each level specified by *-i* or *-fl* or *-e*, in case *-fl* is used alone at least two fixed levels must be specified.

The minimum and maximum values from the raster are not automatically added to the fixed levels list but the special values `MIN` and `MAX` (case insensitive) can be used to include them.

-gt <n>

Group n features per transaction (default 100 000). Increase the value for better performance when writing into DBMS drivers that have transaction support. `n` can be set to unlimited to load the data into a single transaction. If set to 0, no explicit transaction is done.

Added in version 3.10.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

C API

Functionality of this utility can be done from C with *GDALContourGenerate()*.

Examples

Example 1

title
Creating contours from a DEM

```
gdal_contour -a elev dem.tif contour.shp -i 10.0
```

This would create 10-meter contours from the DEM data in `dem.tif` and produce a shapefile in `contour.shp|shx|dbf` with the contour elevations in the `elev` attribute.

Example 2

title
Creating polygonal contours from a DEM

```
$ cat test.asc
ncols        2
nrows        2
xllcorner    0
yllcorner    0
cellsize     1
4 15
25 36

$ gdal_contour test.asc -f GeoJSON /vsistdout/ -i 10 -p -amin min -amax max
```

This would create 10-meter polygonal contours from the DEM data in `test.asc` and produce a GeoJSON output with the contour min and max elevations in the `min` and `max` attributes, including the minimum and maximum values from the raster.

```
{
    "type": "FeatureCollection",
    "name": "contour",
    "features": [
    { "type": "Feature", "properties": { "ID": 0, "min": 4.0, "max": 10.0 }, "geometry":
↪{ "type": "MultiPolygon", "coordinates": [ [ [ [ 0.5, 1.214285714285714 ], [ 1.
↪045454545454545, 1.5 ], [ 1.045454545454545, 2.0 ], [ 1.0, 2.0 ], [ 0.5, 2.0 ], [ 0.0,␣
↪2.0 ], [ 0.0, 1.5 ], [ 0.0, 1.214285714285714 ], [ 0.5, 1.214285714285714 ] ] ] ] } },
    { "type": "Feature", "properties": { "ID": 1, "min": 10.0, "max": 20.0 }, "geometry
↪": { "type": "MultiPolygon", "coordinates": [ [ [ [ 1.5, 1.261904761904762 ], [ 2.0, 1.
↪261904761904762 ], [ 2.0, 1.5 ], [ 2.0, 2.0 ], [ 1.5, 2.0 ], [ 1.045454545454545, 2.0␣
↪], [ 1.045454545454545, 1.5 ], [ 0.5, 1.214285714285714 ], [ 0.0, 1.214285714285714 ],␣
↪[ 0.0, 1.0 ], [ 0.0, 0.738095238095238 ], [ 0.5, 0.738095238095238 ], [ 1.5, 1.
```

(continues on next page)

(continued from previous page)

```
↪261904761904762 ] ] ] ] } },
    { "type": "Feature", "properties": { "ID": 2, "min": 20.0, "max": 30.0 }, "geometry
↪": { "type": "MultiPolygon", "coordinates": [ [ [ [ 0.954545454545455, 0.0 ], [ 0.
↪954545454545455, 0.5 ], [ 1.5, 0.785714285714286 ], [ 2.0, 0.785714285714286 ], [ 2.0,␣
↪1.0 ], [ 2.0, 1.261904761904762 ], [ 1.5, 1.261904761904762 ], [ 0.5, 0.
↪738095238095238 ], [ 0.0, 0.738095238095238 ], [ 0.0, 0.5 ], [ 0.0, 0.0 ], [ 0.5, 0.0␣
↪], [ 0.954545454545455, 0.0 ] ] ] ] } },
    { "type": "Feature", "properties": { "ID": 3, "min": 30.0, "max": 36.0 }, "geometry
↪": { "type": "MultiPolygon", "coordinates": [ [ [ [ 1.499999909090926, 0.0 ], [ 1.0, 0.
↪0 ], [ 0.954545454545455, 0.0 ], [ 0.954545454545455, 0.5 ], [ 1.5, 0.785714285714286␣
↪], [ 2.0, 0.785714285714286 ], [ 2.0, 0.500000047619043 ], [ 1.5, 0.500000047619043 ],␣
↪[ 1.499999909090926, 0.5 ], [ 1.499999909090926, 0.0 ] ] ] ] } } }
    ]
}
```

Example 3

title
Creating contours from a DEM with fixed levels

```
$ cat test.asc
ncols        2
nrows        2
xllcorner    0
yllcorner    0
cellsize     1
4 15
25 36

$ gdal_contour test.asc -f GeoJSON /vsistdout/ -fl 10 20 -p -amin min -amax max
```

This would create a single polygonal contour between 10 and 20 meters from the DEM data in `test.asc` and produce a GeoJSON output with the contour min and max elevations in the `min` and `max` attributes.

If the minimum and maximum values from the raster are desired, the special values `MIN` and `MAX` (case insensitive) can be used:

```
$ cat test.asc
ncols        2
nrows        2
xllcorner    0
yllcorner    0
cellsize     1
4 15
25 36

$ gdal_contour test.asc -f GeoJSON /vsistdout/ -fl MIN 10 20 MAX -p -amin min -amax max
```

This would create three polygonal contours from the DEM data in `test.asc` and produce a GeoJSON output with the contour min and max elevations in the `min` and `max` attributes, the values of these fields will be: (4.0, 10.0), (10, 20.0) and (20.0, 36.0).

Example 4

title
Creating contours from a DEM specifying an interval and fixed levels at the same time

```
$ cat test.asc
ncols        2
nrows        2
xllcorner    0
yllcorner    0
cellsize     1
4 15
25 36

$ gdal_contour test.asc -f GeoJSON /vsistdout/ -i 10 -fl 15 -p -amin min -amax max
```

Creates contours at regular 10 meter intervals and adds extra contour for a fixed 15 m level. Finally turns areas between the contours into polygons with the contour min and max elevations in the `min` and `max` attributes, the values of these fields will be: (4.0, 10.0), (10, 15.0), (15, 20.0), (20.0, 30.0) and (30.0, 36.0).

5.2.2.7 gdal_create

Synopsis

```
Usage: gdal_create [--help] [--long-usage] [--help-general]
                   [-ot Byte|Int8|[U]Int{16|32|64}|CInt{16|32}|[C]Float{32|64}]
                   [-of <output_format>] [-outsize <xsize> <ysize>]
                   [-bands <count>] [-burn <value>]... [-a_srs <srs_def>]
                   [-a_ullr <ulx> <uly> <lrx> <lry>] [-a_nodata <value>]
                   [-mo <NAME>=<VALUE>]... [-co <NAME>=<VALUE>]... [--quiet]
                   [-if <input_dataset>]
                   <out_dataset>
```

Description

The **gdal_create** utility can be used to initialize a new raster file, from its dimensions, band count and set various parameters, such as CRS, geotransform, nodata value, metadata. It can be used also in special cases, like creating a PDF file from a XML composition file.

> **Tip**
>
> Equivalent in new "gdal" command line interface:
>
> See *gdal raster create*.

--help
Show this help message and exit

--help-general
Gives a brief usage message for the generic GDAL commandline options and exit.

-q, --quiet
Suppress progress messages and other non-error output.

-ot <type>

Force the output image bands to have a specific data type supported by the driver, which may be one of the following: `Byte`, `Int8`, `UInt16`, `Int16`, `UInt32`, `Int32`, `UInt64`, `Int64`, `Float32`, `Float64`, `CInt16`, `CInt32`, `CFloat32` or `CFloat64`.

-of <format>

Select the output format. If not specified, the format is guessed from the extension. Use the short format name.

-outsize <xsize> <ysize>

Set the size of the output file in pixels. First value is width. Second one is height.

-bands <count>

Number of bands. Defaults to 1 if -outsize is specified, or 0 otherwise.

-burn <value>

A fixed value to burn into a band. A list of *-burn* options can be supplied, one per band (the first value will apply to the first band, the second one to the second band, etc.). If a single value is specified, it will apply to all bands.

-a_srs <srs_def>

Override the projection for the output file. The<srs_def> may be any of the usual GDAL/OGR forms, complete WKT, PROJ.4, EPSG:n or a file containing the WKT. No reprojection is done.

-a_ullr <ulx> <uly> <lrx> <lry>

Assign the georeferenced bounds of the output file.

-a_nodata <value>

Assign a specified nodata value to output bands.

-mo <META-TAG>=<VALUE>

Passes a metadata key and value to set on the output dataset if possible.

-co <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--format* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

-if <input_dataset>

Added in version 3.3.

Name of GDAL input dataset that serves as a template for default values of options -outsize, -bands, -ot, -a_srs, -a_ullr and -a_nodata. Note that the pixel values will *not* be copied.

<out_dataset>

The destination file name.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Initialize a new GeoTIFF file with a uniform value of 10

```
gdal_create -outsize 20 20 -a_srs EPSG:4326 -a_ullr 2 50 3 49 -burn 10 out.tif
```

Example 2: Create a PDF file from a XML composition file

```
gdal_create -co COMPOSITION_FILE=composition.xml out.pdf
```

Example 3: Initialize a blank GeoTIFF file from an input one

```
gdal_create -if prototype.tif output.tif
```

5.2.2.8 gdal_edit

Synopsis

```
gdal_edit [--help] [--help-general] [-ro] [-a_srs <srs_def>]
        [-a_ullr <ulx> <uly> <lrx> <lry>] [-a_ulurll <ulx> <uly> <urx> <ury> <llx> <lly>]
        [-tr <xres> <yres>] [-unsetgt] [-unsetrpc] [-a_nodata <value>] [-unsetnodata]
        [-a_coord_epoch <epoch>] [-unsetepoch]
        [-unsetstats] [-stats] [-approx_stats]
        [-setstats <min> <max> <mean> <stddev>]
        [-scale <value>] [-offset <value>] [-units <value>]
        [-colorinterp_<X>
↪{red|green|blue|alpha|gray|undefined|pan|coastal|rededge|nir|swir|mwir|lwir|...}]...
        [-gcp <pixel> <line> <easting> <northing> [<elevation>]]...
        [-unsetmd] [-oo <NAME>=<VALUE>]... [-mo <META-TAG>=<VALUE>]...
        <datasetname>
```

Description

gdal_edit can be used to edit in place various information of an existing GDAL dataset (projection, geotransform, nodata, metadata).

It works only with raster formats that support update access to existing datasets.

> **Caution**
>
> Depending on the format, older values of the updated information might still be found in the file in a "ghost" state, even if no longer accessible through the GDAL API. This is for example the case of the *GTiff -- GeoTIFF File Format* format (this is not a exhaustive list).

> **Note**
>
> gdal_edit is a Python utility, and is only available if GDAL Python bindings are available.

Tip

Equivalent in new "gdal" command line interface:

See *gdal raster edit*.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-ro

Open the dataset in read-only. Might be useful for drivers refusing to use the dataset in update-mode. In which case, updated information will go into PAM `.aux.xml` files.

-a_srs <srs_def>

Defines the target coordinate system. This coordinate system will be written to the dataset. If the empty string or None is specified, then the existing coordinate system will be removed (for TIFF/GeoTIFF, might not be well supported besides that).

-a_coord_epoch <epoch>

Assign/override the coordinate epoch of the dataset, as a decimal year (e.g., 2021.3).

Added in version 3.9.

-unsetepoch

Remove the coordinate epoch of the dataset.

Added in version 3.9.

-a_ullr <ulx> <uly> <lrx> <lry>

Assign/override the georeferenced bounds of the dataset.

-a_ulurll <ulx> <uly> <urx> <ury> <llx> <lly>

Assign/override the georeferenced bounds of the dataset from three points: upper-left, upper-right and lower-left. Unlike *-a_ullr*, this also supports rotated datasets (edges not parallel to coordinate system axes).

Added in version 3.1.

-tr <xres> <yres>

Set target resolution. The values must be expressed in georeferenced units. Both must be positive values.

-unsetgt

Remove the georeference information.

-unsetrpc

Remove RPC information.

-unsetstats

Remove band statistics information.

-stats

Calculate and store band statistics.

-setstats <min> <max> <mean> <stddev>

Store user-defined values for band statistics (minimum, maximum, mean and standard deviation). If any of the values is set to None, the real statistics are calculated from the file and the ones set to None are used from the real statistics.

-approx_stats

Calculate and store approximate band statistics.

-a_nodata <value>

Assign a specified nodata value to output bands.

-unsetnodata

Remove existing nodata values.

-scale <value>

Assign a specified scale value to output bands. If a single scale value is provided it will be set for all bands. Alternatively one scale value per band can be provided, in which case the number of scale values must match the number of bands. If no scale is needed, it it recommended to set the value to 1. Scale and Offset are generally used together. For example, scale and offset might be used to store elevations in a unsigned 16bit integer file with a precision of 0.1, and starting from -100. True values would be calculated as: true_value = (pixel_value * scale) + offset

> **Note**
>
> These values can be applied using -unscale during a **gdal_translate** run.

-offset <value>

Assign a specified offset value to output bands. If a single offset value is provided it will be set for all bands. Alternatively one offset value per band can be provided, in which case the number of offset values must match the number of bands. If no offset is needed, it recommended to set the value to 0. For more see scale.

-units <value>

Assign a unit to output band(s).

Added in version 3.1.

-colorinterp_<X> {red|green|blue|alpha|gray|undefined|pan|coastal|rededge|nir|swir|mwir|lwir|...}

Change the color interpretation of band X (where X is a valid band number, starting at 1).

-gcp pixel <line> <easting> <northing> [<elevation>]

Add the indicated ground control point to the dataset. This option may be provided multiple times to provide a set of GCPs.

-unsetmd

Remove existing metadata (in the default metadata domain). Can be combined with *-mo*.

-mo <META-TAG>=<VALUE>

Passes a metadata key and value to set on the output dataset if possible. This metadata is added to the existing metadata items, unless *-unsetmd* is also specified.

-oo <NAME>=<VALUE>

Open option (format specific).

The *-a_ullr*, *-a_ulurll*, *-tr* and *-unsetgt* options are exclusive.

The *-unsetstats* and either *-stats* or *-approx_stats* options are exclusive.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Example

Example 1: Override the spatial bounds of a dataset and assign metadata values

```
gdal_edit -mo DATUM=WGS84 -mo PROJ=GEODETIC -a_ullr 7 47 8 46 test.ecw
```

Example 2: Assign scale and offset values to dataset

```
gdal_edit -scale 1e3 1e4 -offset 0 10 twoBand.tif
```

The values in the GeoTIFF are not changed.

5.2.2.9 gdal_fillnodata

Synopsis

```
gdal_fillnodata [--help] [--help-general] [-q] [-md <max_distance>]
           [-si <smoothing_iterations>] [-o <name>=<value> [<name>=<value> ...]]
           [-mask <filename>] [-interp {inv_dist,nearest}] [-b <band>]
           [-of <gdal_format>] [-co <name>=<value>]
           <src_file> <dst_file>
```

Description

gdal_fillnodata fills selection regions (usually nodata areas) by interpolating from valid pixels around the edges of the area.

Additional details on the algorithm are available in the *GDALFillNodata()* docs.

> **Tip**
>
> Equivalent in new "gdal" command line interface:
>
> See *gdal raster fill-nodata*.

> **Note**
>
> gdal_fillnodata is a Python utility, and is only available if GDAL Python bindings are available.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-q

The script runs in quiet mode. The progress monitor is suppressed and routine messages are not displayed.

-md <max_distance>

The maximum distance (in pixels) that the algorithm will search out for values to interpolate. The default is 100 pixels.

-si <smooth_iterations>

The number of 3x3 average filter smoothing iterations to run after the interpolation to dampen artifacts. The default is zero smoothing iterations.

-o <name>=<value>

Specify a special argument to the algorithm. Currently none are supported.

-b <band>

The band to operate on, by default the first band is operated on.

-mask <filename>

Use the first band of the specified file as a validity mask (zero is invalid, non-zero is valid).

-of <format>

Select the output format. The default is *GTiff -- GeoTIFF File Format*. Use the short format name.

-interp {inv_dist,nearest}

Added in version 3.9.

By default, pixels are interpolated using an inverse distance weighting (`inv_dist`). It is also possible to choose a nearest neighbour (`nearest`) strategy.

<srcfile>

The source raster file used to identify target pixels. Only one band is used.

<dstfile>

The new file to create with the interpolated result.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

5.2.2.10 gdal_footprint

Synopsis

```
Usage: gdal_footprint [--help] [--long-usage] [--help-general]
                      [-b <band>]... [-combine_bands VAR]
                      [[-ovr <index>]|[-srcnodata "<value>[ <value>]..."]]
                      [-t_cs VAR] [-t_srs <srs_def>] [-split_polys]
                      [-convex_hull] [-densify <value>] [-simplify <value>]
                      [-min_ring_area <value>] [-max_points <value>|unlimited]
                      [--quiet] [-oo <NAME>=<VALUE>]... [-of <output_format>]
                      [[-location_field_name <field_name>]|[-no_location]]
                      [-write_absolute_path] [-lco <NAME>=<VALUE>]...
                      [-dsco <NAME>=<VALUE>]... [-lyr_name <value>] [-overwrite]
                      <src_filename> <dst_filename>
```

Description

The **gdal_footprint** utility can be used to compute the footprint of a raster file, taking into account nodata values (or more generally the mask band attached to the raster bands), and generating polygons/multipolygons corresponding to areas where pixels are valid, and write to an output vector file.

The **nearblack** utility may be run as a pre-processing step to generate proper mask bands.

> **Tip**
>
> Equivalent in new "gdal" command line interface:
>
> See *gdal raster footprint*.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-q, --quiet

Suppress progress messages and other non-error output.

-b <band>

Band(s) of interest. Between 1 and the number of bands of the raster. May be specified multiple times. If not specified, all bands are taken into account. The way multiple bands are combined is controlled by *-combine_bands*

-combine_bands `union|intersection`

Defines how the mask bands of the selected bands are combined to generate a single mask band, before being vectorized. The default value is `union`: that is a pixel is valid if it is valid at least for one of the selected bands. `intersection` means that a pixel is valid only if it is valid for all selected bands.

-ovr <index>

To specify which overview level of source file must be used, when overviews are available on the source raster. By default the full resolution level is used. The index is 0-based, that is 0 means the first overview level. This option is mutually exclusive with *-srcnodata*.

-srcnodata "<value>[<value>]..."

Set nodata values for input bands (different values can be supplied for each band). If a single value is specified, it applies to all selected bands. If more than one value is supplied, there should be as many values as the number of selected bands, and all values should be quoted to keep them together as a single operating system argument. If the option is not specified, the intrinsic mask band of each selected bands will be used.

-t_cs `pixel|georef`

Target coordinate system. By default if the input dataset is georeferenced, `georef` is implied, that is the footprint geometry will be expressed as coordinates in the CRS of the raster (or the one specified with *-t_srs*). If specifying `pixel`, the coordinates of the footprint geometry are column and line indices.

-t_srs <srs_def>

Target CRS of the output file. The <srs_def> may be any of the usual GDAL/OGR forms, complete WKT, PROJ.4, EPSG:n or a file containing the WKT. Specifying this option implies -t_cs georef The footprint is reprojected from the CRS of the source raster to the specified CRS.

-split_polys

When specified, multipolygons are split as several features each with one single polygon.

-convex_hull

When specified, the convex hull of (multi)polygons is computed.

-densify <value>

The specified value of this option is the maximum distance between 2 consecutive points of the output geometry. The unit of the distance is in pixels if *-t_cs* equals `pixel`, or otherwise in georeferenced units of the source raster. This option is applied before the reprojection implied by *-t_srs*.

-simplify <value>

The specified value of this option is the tolerance used to merge consecutive points of the output geometry using the *OGRGeometry::Simplify()* method. The unit of the distance is in pixels if *-t_cs* equals `pixel`, or otherwise in georeferenced units of the target vector dataset. This option is applied after the reprojection implied by *-t_srs*.

-min_ring_area <value>

Minimum value for the area of a ring The unit of the area is in square pixels if *-t_cs* equals `pixel`, or otherwise in georeferenced units of the target vector dataset. This option is applied after the reprojection implied by *-t_srs*

-max_points <value>|`unlimited`

Maximum number of points of each output geometry (not counting the closing point of each ring, which is always identical to the first point). The default value is 100. `unlimited` can be used to remove that limitation.

-oo <NAME>=<VALUE>

Dataset open option (format specific)

-of <ogr_format>

Select the output format. Use the short format name. Guessed from the file extension if not specified

-location_field_name <field_name>

Added in version 3.9.0.

Specifies the name of the field in the resulting vector dataset where the path of the input dataset will be stored. The default field name is "location". To prevent writing the path of the input dataset, use *-no_location*

-no_location

Added in version 3.9.0.

Turns off the writing of the path of the input dataset as a field in the output vector dataset.

-write_absolute_path

Added in version 3.9.0.

Enables writing the absolute path of the input dataset. By default, the filename is written in the location field exactly as specified on the command line.

-lco <NAME>=<VALUE>

Layer creation option (format specific)

-dsco <NAME>=<VALUE>

Dataset creation option (format specific)

-lyr_name <value>

Name of the target layer. `footprint` if not specified.

-overwrite

Overwrite the target layer if it exists.

<src_filename>

The source raster file name.

<dst_filename>

The destination vector file name. If the file and the output layer exist, the new footprint is appended to them, unless *-overwrite* is used.

Post-vectorization geometric operations are applied in the following order:

- optional splitting (*-split_polys*)
- optional densification (*-densify*)
- optional reprojection (*-t_srs*)
- optional filtering by minimum ring area (*-min_ring_area*)
- optional application of convex hull (*-convex_hull*)
- optional simplification (*-simplify*)
- limitation of number of points (*-max_points*)

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

C API

This utility is also callable from C with *GDALFootprint()*.

Examples

Example 1: Compute the footprint of a GeoTIFF file as a GeoJSON file

```
gdal_footprint -t_srs EPSG:4326 input.tif output.geojson
```

The footprint will be written using WGS84 longitude, latitude coordinates, regardless of the spatial reference system used by the input raster.

5.2.2.11 gdal_grid

Synopsis

```
gdal_grid [--help] [--help-general]
          [-ot {Byte/Int16/UInt16/UInt32/Int32/Float32/Float64/
          CInt16/CInt32/CFloat32/CFloat64}]
          [-oo <NAME>=<VALUE>]...
          [-of <format>] [-co <NAME>=<VALUE>]...
          [-zfield <field_name>] [-z_increase <increase_value>] [-z_multiply <multiply_
↪value>]
          [-a_srs <srs_def>] [-spat <xmin> <ymin> <xmax> <ymax>]
          [-clipsrc <xmin> <ymin> <xmax> <ymax>|<WKT>|<datasource>|spat_extent]
          [-clipsrcsql <sql_statement>] [-clipsrclayer <layer>]
          [-clipsrcwhere <expression>]
          [-l <layername>]... [-where <expression>] [-sql <select_statement>]
          [-txe <xmin> <xmax>] [-tye <ymin> <ymax>] [-tr <xres> <yres>] [-outsize <xsize>
↪ <ysize>]
          [-a {<algorithm>[[:<parameter1>=<value1>]...]}] [-q]
          <src_datasource> <dst_filename>
```

Description

This program creates a regular grid (raster) from the scattered data read from the OGR datasource. Input data will be interpolated to fill grid nodes with values, you can choose from various interpolation methods.

It is possible to set the *GDAL_NUM_THREADS* configuration option to parallelize the processing. The value to specify is the number of worker threads, or `ALL_CPUS` to use all the cores/CPUs of the computer.

> **Tip**
>
> Equivalent in new "gdal" command line interface:
>
> See *gdal vector grid*.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-q, --quiet

Suppress progress messages and other non-error output.

-ot <type>

Force the output image bands to have a specific data type supported by the driver, which may be one of the following: `Byte`, `Int8`, `UInt16`, `Int16`, `UInt32`, `Int32`, `UInt64`, `Int64`, `Float32`, `Float64`, `CInt16`, `CInt32`, `CFloat32` or `CFloat64`.

If not set then a default type is used, which might not be supported by the relevant driver, causing a error.

-of <format>

Select the output format. If not specified, the format is guessed from the extension. Use the short format name.

-txe <xmin> <xmax>

Set georeferenced X extents of output file to be created.

-tye <ymin> <ymax>

Set georeferenced Y extents of output file to be created.

-tr <xres> <yres>

Set output file resolution (in target georeferenced units). Note that *-tr* just works in combination with a valid input from *-txe* and *-tye*

Added in version 3.2.

-outsize <xsize> <ysize>

Set the size of the output file in pixels and lines. Note that *-outsize* cannot be used with *-tr*

-a_srs <srs_def>

Override the projection for the output file. The *srs_def* may be any of the usual GDAL/OGR forms, complete WKT, PROJ.4, EPSG:n or a file containing the WKT. No reprojection is done.

-zfield <field_name>

Identifies an attribute field on the features to be used to get a Z value from. This value overrides the Z value read from the feature geometry record (naturally, if you have a Z value in the geometry, otherwise you have no choice and should specify a field name containing a Z value).

-z_increase <increase_value>

Addition to the attribute field on the features to be used to get a Z value from. The addition should be the same unit as the Z value. The result value will be Z value + Z increase value. The default value is 0.

-z_multiply <multiply_value>

This is multiplication ratio for the Z field. This can be used for a shift from e.g. feet to meters or from elevation to depth. The result value will be (Z value + Z increase value) * Z multiply value. The default value is 1.

-a {<algorithm>[[:<parameter1>=<value1>]...]}

Set the interpolation algorithm or data metric name and (optionally) its parameters. See the *Interpolation algorithms* and *Data metrics* sections for further discussion of available options.

-spat <xmin> <ymin> <xmax> <ymax>

Adds a spatial filter to select only features contained within the bounding box described by (xmin, ymin) - (xmax, ymax).

-clipsrc [<xmin> <ymin> <xmax> <ymax>]|<WKT>|<datasource>|spat_extent

Adds a spatial filter to select only features contained within the specified bounding box (expressed in source SRS), WKT geometry (POLYGON or MULTIPOLYGON), from a datasource or to the spatial extent of the *-spat* option if you use the `spat_extent` keyword. When specifying a datasource, you will generally want to use it in combination with the *-clipsrclayer*, *-clipsrcwhere* or *-clipsrcsql* options.

-clipsrcsql <sql_statement>

Select desired geometries using an SQL query instead.

-clipsrclayer <layername>

Select the named layer from the source clip datasource.

-clipsrcwhere <expression>

Restrict desired geometries based on an attribute query.

-l <layername>

Indicates the layer(s) from the datasource that will be used for input features. May be specified multiple times, but at least one layer name or a *-sql* option must be specified.

-where <expression>

An optional SQL WHERE style query expression to be applied to select features to process from the input layer(s).

-sql <select_statement>

An SQL statement to be evaluated against the datasource to produce a virtual layer of features to be processed.

-oo <NAME>=<VALUE>

Added in version 3.7.

Source dataset open option (format specific)

-co <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--format* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

<src_datasource>

Any OGR supported readable datasource.

`<dst_filename>`

The GDAL supported output file.

Interpolation algorithms

There are a number of interpolation algorithms to choose from.

More details about them can also be found in *GDAL Grid Tutorial*

invdist

Inverse distance to a power. This is the default algorithm. It has the following parameters:

- `power`: Weighting power (default 2.0).
- `smoothing`: Smoothing parameter (default 0.0).
- `radius1`: The first radius (X axis if rotation angle is 0) of the search ellipse. Set this parameter to zero to use the whole point array. Default is 0.0.
- `radius2`: The second radius (Y axis if rotation angle is 0) of the search ellipse. Set this parameter to zero to use the whole point array. Default is 0.0.
- `radius`: Set first and second radius (mutually exclusive with radius1 and radius2). Default is 0.0. Added in GDAL 3.6
- `angle`: Angle of search ellipse rotation in degrees (counter clockwise, default 0.0).
- `max_points`: Maximum number of data points to use. Do not search for more points than this number. This is only used if the search ellipse is set (both radii are non-zero). Zero means that all found points should be used. Default is 0.
- `min_points`: Minimum number of data points to use. If less amount of points found the grid node considered empty and will be filled with NODATA marker. This is only used if search ellipse is set (both radii are non-zero). Default is 0.
- `max_points_per_quadrant`: Maximum number of data points to use per quadrant. Default is 0. Added in GDAL 3.6. When specified, this actually uses invdistnn implementation.
- `min_points_per_quadrant`: Minimum number of data points to use per quadrant. Default is 0. Added in GDAL 3.6. When specified, this actually uses invdistnn implementation.
- `nodata`: NODATA marker to fill empty points (default 0.0).

invdistnn

Inverse distance to a power with nearest neighbor searching, ideal when max_points is used. It has following parameters:

- `power`: Weighting power (default 2.0).
- `smoothing`: Smoothing parameter (default 0.0).
- `radius`: The radius of the search circle, which should be non-zero. Default is 1.0.
- `max_points`: Maximum number of data points to use. Do not search for more points than this number. Found points will be ranked from nearest to furthest distance when weighting. Default is 12.
- `min_points`: Minimum number of data points to use. If less amount of points found the grid node is considered empty and will be filled with NODATA marker. Default is 0.
- `max_points_per_quadrant`: Maximum number of data points to use per quadrant. Default is 0. Added in GDAL 3.6. When specified, the algorithm will only take into account up to max_points_per_quadrant points for each of the right-top, left-top, right-bottom and right-top quadrant relative to the point being interpolated.

- `min_points_per_quadrant`: Minimum number of data points to use per quadrant. Default is 0. Added in GDAL 3.6. If that number is not reached, the point being interpolated will be set with the NODATA marker. When specified, the algorithm will collect at least min_points_per_quadrant points for each of the right-top, left-top, right-bottom and right-top quadrant relative to the point being interpolated.
- `nodata`: NODATA marker to fill empty points (default 0.0).

When `min_points_per_quadrant` or `max_points_per_quadrant` is specified, the search will start with the closest point to the point being interpolated from the first quadrant, then the closest point to the point being interpolated from the second quadrant, etc. up to the 4th quadrant, and will continue with the next closest point in the first quadrant, etc. until `max_points` and/or `max_points_per_quadrant` thresholds are reached.

average

Moving average algorithm. It has following parameters:

- `radius1`: The first radius (X axis if rotation angle is 0) of search ellipse. Set this parameter to zero to use whole point array. Default is 0.0.
- `radius2`: The second radius (Y axis if rotation angle is 0) of search ellipse. Set this parameter to zero to use whole point array. Default is 0.0.
- `radius`: Set first and second radius (mutually exclusive with radius1 and radius2). Default is 0.0. Added in GDAL 3.6
- `angle`: Angle of search ellipse rotation in degrees (counter clockwise, default 0.0).
- `max_points`: Maximum number of data points to use. Do not search for more points than this number. Found points will be ranked from nearest to furthest distance when weighting. Default is 0. Added in GDAL 3.6 Only taken into account if one or both of `min_points_per_quadrant` or `max_points_per_quadrant` is specified
- `min_points`: Minimum number of data points to use. If less amount of points found the grid node considered empty and will be filled with NODATA marker. Default is 0.
- `max_points_per_quadrant`: Maximum number of data points to use per quadrant. Default is 0. Added in GDAL 3.6. When specified, the algorithm will only take into account up to max_points_per_quadrant points for each of the right-top, left-top, right-bottom and right-top quadrant relative to the point being interpolated.
- `min_points_per_quadrant`: Minimum number of data points to use per quadrant. Default is 0. Added in GDAL 3.6. If that number is not reached, the point being interpolated will be set with the NODATA marker. When specified, the algorithm will collect at least min_points_per_quadrant points for each of the right-top, left-top, right-bottom and right-top quadrant relative to the point being interpolated.
- `nodata`: NODATA marker to fill empty points (default 0.0).

Note, that it is essential to set search ellipse for moving average method. It is a window that will be averaged when computing grid nodes values.

When `min_points_per_quadrant` or `max_points_per_quadrant` is specified, the search will start with the closest point to the point being interpolated from the first quadrant, then the closest point to the point being interpolated from the second quadrant, etc. up to the 4th quadrant, and will continue with the next closest point in the first quadrant, etc. until `max_points` and/or `max_points_per_quadrant` thresholds are reached.

nearest

Nearest neighbor algorithm. It has following parameters:

- `radius1`: The first radius (X axis if rotation angle is 0) of search ellipse. Set this parameter to zero to use whole point array. Default is 0.0.

- `radius2`: The second radius (Y axis if rotation angle is 0) of search ellipse. Set this parameter to zero to use whole point array. Default is 0.0.
- `radius`: Set first and second radius (mutually exclusive with radius1 and radius2). Default is 0.0. Added in GDAL 3.6
- `angle`: Angle of search ellipse rotation in degrees (counter clockwise, default 0.0).
- `nodata`: NODATA marker to fill empty points (default 0.0).

linear

Linear interpolation algorithm.

The Linear method performs linear interpolation by computing a Delaunay triangulation of the point cloud, finding in which triangle of the triangulation the point is, and by doing linear interpolation from its barycentric coordinates within the triangle. If the point is not in any triangle, depending on the radius, the algorithm will use the value of the nearest point or the nodata value.

It has following parameters:

- `radius`: In case the point to be interpolated does not fit into a triangle of the Delaunay triangulation, use that maximum distance to search a nearest neighbour, or use nodata otherwise. If set to -1, the search distance is infinite. If set to 0, nodata value will be always used. Default is -1.
- `nodata`: NODATA marker to fill empty points (default 0.0).

Data metrics

Besides the interpolation functionality **`gdal_grid`** can be used to compute some data metrics using the specified window and output grid geometry. These metrics are:

- `minimum`: Minimum value found in grid node search ellipse.
- `maximum`: Maximum value found in grid node search ellipse.
- `range`: A difference between the minimum and maximum values found in grid node search ellipse.
- `count`: A number of data points found in grid node search ellipse.
- `average_distance`: An average distance between the grid node (center of the search ellipse) and all of the data points found in grid node search ellipse.
- `average_distance_pts`: An average distance between the data points found in grid node search ellipse. The distance between each pair of points within ellipse is calculated and average of all distances is set as a grid node value.

All the metrics have the same set of options:

- `radius1`: The first radius (X axis if rotation angle is 0) of search ellipse. Set this parameter to zero to use whole point array. Default is 0.0.
- `radius2`: The second radius (Y axis if rotation angle is 0) of search ellipse. Set this parameter to zero to use whole point array. Default is 0.0.
- `radius`: Set first and second radius (mutually exclusive with radius1 and radius2). Default is 0.0. Added in GDAL 3.6
- `angle`: Angle of search ellipse rotation in degrees (counter clockwise, default 0.0).
- `min_points`: Minimum number of data points to use. If less amount of points found the grid node considered empty and will be filled with NODATA marker. This is only used if search ellipse is set (both radii are non-zero). Default is 0.

- `max_points_per_quadrant`: Maximum number of data points to use per quadrant. Default is 0. Added in GDAL 3.6. When specified, the algorithm will only take into account up to max_points_per_quadrant points for each of the right-top, left-top, right-bottom and right-top quadrant relative to the point being interpolated.
- `min_points_per_quadrant`: Minimum number of data points to use per quadrant. Default is 0. Added in GDAL 3.6. If that number is not reached, the point being interpolated will be set with the NODATA marker. When specified, the algorithm will collect at least min_points_per_quadrant points for each of the right-top, left-top, right-bottom and right-top quadrant relative to the point being interpolated.
- `nodata`: NODATA marker to fill empty points (default 0.0).

When `min_points_per_quadrant` or `max_points_per_quadrant` is specified, the search will start with the closest point to the point being interpolated from the first quadrant, then the closest point to the point being interpolated from the second quadrant, etc. up to the 4th quadrant, and will continue with the next closest point in the first quadrant, etc. until `max_points` and/or `max_points_per_quadrant` thresholds are reached.

Reading comma separated values

Often you have a text file with a list of comma separated XYZ values to work with (so called CSV file). You can easily use that kind of data source in **gdal_grid**. All you need is to create a virtual dataset header (VRT) for your CSV file and use it as an input datasource for **gdal_grid**. You can find details on the VRT format on the *VRT -- OGR Virtual Format* description page.

Here is a small example. Let's say we have a CSV file called *dem.csv* containing

```
Easting,Northing,Elevation
86943.4,891957,139.13
87124.3,892075,135.01
86962.4,892321,182.04
87077.6,891995,135.01
...
```

For the above data we will create a *dem.vrt* header with the following content:

```
<OGRVRTDataSource>
    <OGRVRTLayer name="dem">
        <SrcDataSource>dem.csv</SrcDataSource>
        <GeometryType>wkbPoint</GeometryType>
        <GeometryField encoding="PointFromColumns" x="Easting" y="Northing" z="Elevation
↪"/>
    </OGRVRTLayer>
</OGRVRTDataSource>
```

This description specifies so called 2.5D geometry with three coordinates X, Y and Z. The Z value will be used for interpolation. Now you can use *dem.vrt* with all OGR programs (start with *ogrinfo* to test that everything works fine). The datasource will contain a single layer called *"dem"* filled with point features constructed from values in the CSV file. Using this technique you can handle CSV files with more than three columns, switch columns, etc. OK, now the final step:

```
gdal_grid dem.vrt demv.tif
```

Or, if we do not wish to use a VRT file:

```
gdal_grid -l dem -oo X_POSSIBLE_NAMES=Easting \
-oo Y_POSSIBLE_NAMES=Northing -zfield Elevation dem.csv dem.tif
```

If your CSV file does not contain column headers then it can be handled in the VRT file in the following way:

```
<GeometryField encoding="PointFromColumns" x="field_1" y="field_2" z="field_3"/>
```

The *Comma Separated Value (.csv)* description page contains details on CSV format supported by GDAL/OGR.

Creating multiband files

Creating multiband files is not directly possible with gdal_grid. One might use gdal_grid multiple times to create one band per file, and then use *gdalbuildvrt* -separate and then *gdal_translate*:

```
gdal_grid ... 1.tif; gdal_grid ... 2.tif; gdal_grid ... 3.tif
gdalbuildvrt -separate 123.vrt 1.tif 2.tif 3.tif
gdal_translate 123.vrt 123.tif
```

Or just use *gdal_merge*, to combine the one-band files into a single one:

```
gdal_grid ... a.tif; gdal_grid ... b.tif; gdal_grid ... c.tif
gdal_merge -separate a.tif b.tif c.tif -o d.tif
```

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

C API

This utility is also callable from C with `GDALGrid()`.

Examples

Example 1: Create a raster from a VRT datasource using inverse distance to a power

```
gdal_grid -a invdist:power=2.0:smoothing=1.0 -txe 85000 89000 -tye 894000 890000 \
    -outsize 400 400 -of GTiff -ot Float64 -l dem dem.vrt dem.tiff
```

This example creates a raster TIFF file from the VRT datasource described in *Reading comma separated values* section using the inverse distance to a power method. Values to interpolate will be read from Z value of geometry record.

Example 2: Read values to interpolate from an attribute field

```
gdal_grid -zfield "Elevation" -a invdist:power=2.0:smoothing=1.0 -txe 85000 89000 \
    -tye 894000 890000 -outsize 400 400 -of GTiff -ot Float64 -l dem dem.vrt \
    dem.tiff --config GDAL_NUM_THREADS ALL_CPUS
```

This command does the same thing as the previous one, but reads values to interpolate from the attribute field specified with **-zfield** option instead of geometry record. So in this case X and Y coordinates are being taken from geometry and Z is being taken from the *"Elevation"* field. The `GDAL_NUM_THREADS` is also set to parallelize the computation.

5.2.2.12 gdal_merge

Synopsis

```
gdal_merge [--help] [--help-general]
               [-o <out_filename>] [-of <out_format>] [-co <NAME>=<VALUE>]...
               [-ps <pixelsize_x> <pixelsize_y>] [-tap] [-separate] [-q] [-v] [-pct]
               [-ul_lr <ulx> <uly> <lrx> <lry>] [-init "<value>[ <value>]..."]
               [-n <nodata_value>] [-a_nodata <output_nodata_value>]
               [-ot <datatype>] [-createonly] <input_file> [<input_file>]...
```

Description

This utility will automatically mosaic a set of images. All the images must be in the same coordinate system and have a matching number of bands, but they may be overlapping, and at different resolutions. In areas of overlap, the last image will be copied over earlier ones. Nodata/transparency values are considered on a band by band level, i.e. a nodata/transparent pixel on one source band will not set a nodata/transparent value on all bands for the target pixel in the resulting raster nor will it overwrite a valid pixel value.

Note

gdal_merge is a Python utility, and is only available if GDAL Python bindings are available.

Tip

Equivalent in new "gdal" command line interface:

See *gdal raster mosaic*.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-o `<out_filename>`

The name of the output file, which will be created if it does not already exist (defaults to "out.tif").

-of `<format>`

Select the output format. If not specified, the format is guessed from the extension. Use the short format name.

-co `<NAME>=<VALUE>`

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--format* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

-ot `<type>`

Force the output image bands to have a specific data type supported by the driver, which may be one of the following: `Byte`, `Int8`, `UInt16`, `Int16`, `UInt32`, `Int32`, `UInt64`, `Int64`, `Float32`, `Float64`, `CInt16`, `CInt32`, `CFloat32` or `CFloat64`.

-ps <pixelsize_x> <pixelsize_y>

Pixel size to be used for the output file. If not specified the resolution of the first input file will be used.

-tap

(target aligned pixels) align the coordinates of the extent of the output file to the values of the -tr, such that the aligned extent includes the minimum extent. Alignment means that xmin / resx, ymin / resy, xmax / resx and ymax / resy are integer values.

-ul_lr <ulx> <uly> <lrx> <lry>

The extents of the output file. If not specified the aggregate extents of all input files will be used.

-q, -quiet

Suppress progress messages.

-v

Generate verbose output of mosaicing operations as they are done.

-separate

Place each input file into a separate band.

-pct

Grab a pseudo-color table from the first input image, and use it for the output. Merging pseudo-colored images this way assumes that all input files use the same color table.

-n <nodata_value>

Ignore pixels from files being merged in with this pixel value.

-a_nodata <output_nodata_value>

Assign a specified nodata value to output bands.

-init <"value(s)">

Pre-initialize the output image bands with these values. However, it is not marked as the nodata value in the output file. If only one value is given, the same value is used in all the bands.

-createonly

The output file is created (and potentially pre-initialized) but no input image data is copied into it.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Creating an image with the pixels in all bands initialized to 255

```
gdal_merge -init 255 -o out.tif in1.tif in2.tif
```

Example 2: Creating an RGB image that shows blue in pixels with no data

```
gdal_merge -init "0 0 255" -o out.tif in1.tif in2.tif
```

The first two bands will be initialized to 0 and the third band will be initialized to 255.

Example 3: Passing a large list of files to gdal_merge

```
ls -1 *.tif > tiff_list.txt
```

A large list of files can be passed to **gdal_merge** by listing them in a text file using the command above on Linux, or:

```
dir /b /s *.tif > tiff_list.txt
```

on Windows. The text file can then be passed to **gdal_merge** using *--optfile*:

```
gdal_merge -o mosaic.tif --optfile tiff_list.txt
```

Example 4: Creating an RGB image by merging 3 different greyscale bands

Conduct "merging by stacking" with the `-separate` flag. Given three greyscale files that cover the same area, you can run:

```
gdal_merge -separate 1.tif 2.tif 3.tif -o rgb.tif
```

This maps `1.tif` to red, `2.tif` to green and `3.tif` to blue.

Example 5: Specifying overlap precedence

The last image in the input line comes out on top of the finished image stack. You might also need to use `-n` to note which value should not be copied into the destination image if it is not already defined as nodata.

```
gdal_merge -o merge.tif -n 0 image1.tif image2.tif image3.tif image4.tif
```

5.2.2.13 gdal_pansharpen

Synopsis

```
gdal_pansharpen [--help] [--help-general]
                <pan_dataset>
                <spectral_dataset>[,band=<num>] [<spectral_dataset>[,band=<num>]]...
                <out_dataset>
                [-of <format>] [-b <band>]... [-w <weight_val>]...
                [-r {nearest|bilinear|cubic|cubicspline|lanczos|average}]
                [-threads {ALL_CPUS|<number>}] [-bitdepth <val>] [-nodata <val>]
                [-spat_adjust {union|intersection|none|nonewithoutwarning}]
                [-co <NAME>=<VALUE>]... [-q]
```

Description

gdal_pansharpen performs a pan-sharpening operation. It can create a "classic" output dataset (such as GeoTIFF), or a VRT dataset describing the pan-sharpening operation.

More details can be found in the *Pansharpened VRT* section.

> **Note**
>
> gdal_pansharpen is a Python utility, and is only available if GDAL Python bindings are available.

> **Tip**
>
> Equivalent in new "gdal" command line interface:
>
> See *gdal raster pansharpen*.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-of <format>

Select the output format. If not specified, the format is guessed from the extension. Use the short format name. `VRT` can also be used.

-b <band>

Select band *band* from the input spectral bands for output. Bands are numbered from 1 in the order spectral bands are specified. Multiple **-b** switches may be used. When no -b switch is used, all input spectral bands are set for output.

-w <weight_val>

Specify a weight for the computation of the pseudo panchromatic value. There must be as many -w switches as input spectral bands.

-r {nearest|bilinear|cubic|cubicspline|lanczos|average}

Select a resampling algorithm. `cubic` is the default.

-threads {ALL_CPUS|<number>}

Specify number of threads to use to do the resampling and pan-sharpening itself. Can be an integer number or `ALL_CPUS`.

-bitdepth <val>

Specify the bit depth of the panchromatic and spectral bands (e.g. 12). If not specified, the NBITS metadata item from the panchromatic band will be used if it exists.

-nodata <val>

Specify nodata value for bands. Used for the resampling and pan-sharpening computation itself. If not set, deduced from the input bands, provided they have a consistent setting.

-spat_adjust {union|intersection|none|nonewithoutwarning}

Select behavior when bands have not the same extent. See *SpatialExtentAdjustment* documentation in *Pansharpened VRT* `union` is the default.

-co <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--format* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

-q

Suppress progress monitor and other non-error output.

<pan_dataset>

Dataset with panchromatic band (first band will be used).

<spectral_dataset>[,band=<num>]

Dataset with one or several spectral bands. If the band option is not specified, all bands of the datasets are taken into account. Otherwise, only the specified (num)th band. The same dataset can be repeated several times.

<out_dataset>

Output dataset

Bands should be in the same projection.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Example

Example 1: With spectral bands in a single dataset

```
gdal_pansharpen panchro.tif multispectral.tif pansharpened_out.tif
```

Example 2: With a few spectral bands from a single dataset, reordered

```
gdal_pansharpen panchro.tif multispectral.tif,band=3 multispectral.tif,band=2␣
↪multispectral.tif,band=1 pansharpened_out.tif
```

Example 3: With spectral bands in several datasets

```
gdal_pansharpen panchro.tif band1.tif band2.tif band3.tif pansharpened_out.tif
```

Example 4: Specifying weights

```
gdal_pansharpen -w 0.7 -w 0.2 -w 0.1 panchro.tif multispectral.tif pansharpened_out.tif
```

Example 5: Specify RGB bands from a RGBNir multispectral dataset while computing the pseudo panchromatic intensity on the 4 RGBNir bands

```
gdal_pansharpen -b 1 -b 2 -b 3 panchro.tif rgbnir.tif pansharpened_out.tif
```

5.2.2.14 gdal_polygonize

Synopsis

```
gdal_polygonize [--help] [--help-general]
                    [-8] [-o <name>=<value>]... [-nomask]
                    [-mask <filename>] <raster_file> [-b <band>]
                    [-q] [-f <ogr_format>] [-lco <name>=<value>]...
                    [-overwrite] <out_file> [<layer>] [<fieldname>]
```

Description

This utility creates vector polygons for all connected regions of pixels in the raster sharing a common pixel value. Each polygon is created with an attribute indicating the pixel value of that polygon. A raster mask may also be provided to determine which pixels are eligible for processing.

The utility will create the output vector datasource if it does not already exist, otherwise it will try to append to an existing one.

The utility is based on the :*GDALPolygonize()* function which has additional details on the algorithm.

> **Note**
>
> gdal_polygonize is a Python utility, and is only available if GDAL Python bindings are available.

> **Tip**
>
> Equivalent in new "gdal" command line interface:
>
> See *gdal raster polygonize*.

--help
: Show this help message and exit

--help-general
: Gives a brief usage message for the generic GDAL commandline options and exit.

-8
: Use 8 connectedness. Default is 4 connectedness.

-nomask
: Do not use the default validity mask for the input band (such as nodata, or alpha masks).

-mask <filename>
: Use the first band of the specified file as a validity mask (zero is invalid, non-zero is valid). If not specified, the default validity mask for the input band (such as nodata, or alpha masks) will be used (unless -nomask is specified)

<raster_file>
: The source raster file from which polygons are derived.

-b <band>
: The band on <raster_file> to build the polygons from. The value can also be set to "mask", to indicate that the mask band of the first band must be used (or "mask,band_number" for the mask of a specified band)

-f <ogr_format>
: Select the output format. If not specified, the format is guessed from the extension. Use the short format name

-o <NAME>=<VALUE>
: Added in version 3.7.

  Polygonize option. See :*GDALPolygonize()* documentation.

-lco <NAME>=<VALUE>
: Added in version 3.7.

  Layer creation option (format specific)

-overwrite

Added in version 3.8.

Overwrite the output layer if it already exists.

<out_file>

The destination vector file to which the polygons will be written.

<layer>

The name of the layer created to hold the polygon features.

<fieldname>

The name of the field to create (defaults to "DN").

-q

The script runs in quiet mode. The progress monitor is suppressed and routine messages are not displayed.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

5.2.2.15 gdal_proximity

Synopsis

```
gdal_proximity [--help] [--help-general]
                  <srcfile> <dstfile> [-srcband <n>] [-dstband <n>]
                  [-of <format>] [-co <name>=<value>]...
                  [-ot Byte/UInt16/UInt32/Float32/etc]
                  [-values <n>,<n>,<n>] [-distunits {PIXEL|GEO}]
                  [-maxdist <n>] [-nodata <n>] [-use_input_nodata {YES|NO}]
                  [-fixed-buf-val <n>]
```

Description

gdal_proximity generates a raster proximity map indicating the distance from the center of each pixel to the center of the nearest pixel identified as a target pixel. Target pixels are those in the source raster for which the raster pixel value is in the set of target pixel values.

> **Note**
>
> gdal_proximity is a Python utility, and is only available if GDAL Python bindings are available.

> **Tip**
>
> Equivalent in new "gdal" command line interface:
>
> See *gdal raster proximity*.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

<srcfile>

The source raster file used to identify target pixels.

<dstfile>

The destination raster file to which the proximity map will be written. It may be a pre-existing file of the same size as srcfile. If it does not exist it will be created.

-srcband <n>

Identifies the band in the source file to use (default is 1).

-dstband <n>

Identifies the band in the destination file to use (default is 1).

-of <format>

Select the output format. If not specified, the format is guessed from the extension. Use the short format name.

-co <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--format* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

-ot <type>

Specify a data type supported by the driver, which may be one of the following: `Byte`, `UInt16`, `Int16`, `UInt32`, `Int32`, `Float32` (default), or `Float64`.

-values <n>,<n>,<n>

A list of target pixel values in the source image to be considered target pixels. If not specified, all non-zero pixels will be considered target pixels.

-distunits {PIXEL|GEO}

Indicate whether distances generated should be in pixel or georeferenced coordinates (default PIXEL).

-maxdist <n>

The maximum distance to be generated. The nodata value will be used for pixels beyond this distance. If a nodata value is not provided, the output band will be queried for its nodata value. If the output band does not have a nodata value, then the value 65535 will be used. Distance is interpreted in pixels unless -distunits GEO is specified.

-nodata <n>

Specify a nodata value to use for the destination proximity raster.

-use_input_nodata {YES|NO}

Indicate whether nodata pixels in the input raster should be nodata in the output raster (default NO).

-fixed-buf-val <n>

Specify a value to be applied to all pixels that are within the -maxdist of target pixels (including the target pixels) instead of a distance value.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

5.2.2.16 gdal_rasterize

Synopsis

```
Usage: gdal_rasterize [--help] [--long-usage] [--help-general]
                      [-b <band>]... [-i] [-at]
                      [[-burn <value>]...|[-a <attribute_name>]|[-3d]]
                      [-add]
                      [[-l <layer_name>]...|[-sql <sql_statement>]]
                      [-where <expression>] [-dialect <sql_dialect>]
                      [-a_nodata <value>] [-init <value>]... [-a_srs <srs_def>]
                      [-to <NAME>=<VALUE>]... [-te <xmin> <ymin> <xmax> <ymax>]
                      [[-tr <xres> <yres>]|[-ts <width> <height>]]
                      [-tap] [-optim AUTO|VECTOR|RASTER] [-co <NAME>=<VALUE>]...
                      [-ot Byte|Int8|[U]Int{16|32|64}|CInt{16|32}|[C]Float{32|64}]
                      [-of <output_format>] [--quiet] [-oo <NAME>=<VALUE>]...
                      <src_datasource> <dst_filename>
```

Description

This program burns vector geometries (points, lines, and polygons) into the raster band(s) of a raster image. Vectors are read from OGR supported vector formats. If the output raster already exists, the affected pixels are updated in-place.

On the fly reprojection of vector data to the coordinate system of the raster data is supported.

> **Tip**
>
> Equivalent in new "gdal" command line interface:
>
> See *gdal vector rasterize*.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-q, --quiet

Suppress progress messages and other non-error output.

-b <band>

The band(s) to burn values into. Multiple -b arguments may be used to burn into a list of bands. The default is to burn into band 1. Not used when creating a new raster.

-i

Invert rasterization. Burn the fixed burn value, or the burn value associated with the first feature into all parts of the image *not* inside the provided polygon.

> **Note**
>
> When the vector features contain a polygon nested within another polygon (like an island in a lake), GDAL must be built against GEOS to get correct results.

-at

Enables the ALL_TOUCHED rasterization option so that all pixels touched by lines or polygons will be updated, not just those on the line render path, or whose center point is within the polygon (behavior is unspecified when the polygon is just touching the pixel center). Defaults to disabled for normal rendering rules.

> **Note**
>
> When this option is enabled, the order of the input features (lines or polygons) can affect the results. When two features touch each other, the last one (i.e. topmost) will determine the burned pixel value at the edge. You may wish to use the *-sql* option to reorder the features (ORDER BY) to achieve a more predictable result.

-burn `<value>`

A fixed value to burn into a band for all objects. A list of *-burn* options can be supplied, one per band being written to.

-a `<attribute_name>`

Identifies an attribute field on the features to be used for a burn-in value. The value will be burned into all output bands.

-3d

Indicates that a burn value should be extracted from the "Z" values of the feature. Works with points and lines (linear interpolation along each segment). For polygons, works properly only if they are flat (same Z value for all vertices).

-add

Instead of burning a new value, this adds the new value to the existing raster. Suitable for heatmaps for instance.

-l `<layername>`

Indicates the layer(s) from the datasource that will be used for input features. May be specified multiple times, but at least one layer name or a *-sql* option must be specified (not both).

-where `<expression>`

An optional SQL WHERE style query expression to be applied to select features to burn in from the input layer(s).

-sql `<select_statement>`

An SQL statement to be evaluated against the datasource to produce a virtual layer of features to be burned in. Starting with GDAL 3.7, the `@filename` syntax can be used to indicate that the content is in the pointed filename.

> **Note**
>
> This option will be ignored if the *-l* option has been set as well.

-dialect `<dialect>`

SQL dialect. In some cases can be used to use (unoptimized) OGR SQL instead of the native SQL of an RDBMS by passing OGRSQL. The "SQLITE" dialect can also be used with any datasource.

-of <format>

Select the output format. If not specified, the format is guessed from the extension. Use the short format name.

-a_nodata <value>

Assign a specified nodata value to output bands.

-init <value>

Pre-initialize the output image bands with these values. However, it is not marked as the nodata value in the output file. If only one value is given, the same value is used in all the bands.

-a_srs <srs_def>

Override the projection for the output file. If not specified, the projection of the input vector file will be used if available. When using this option, no reprojection of features from the SRS of the input vector to the specified SRS of the output raster, so use only this option to correct an invalid source SRS. The <srs_def> may be any of the usual GDAL/OGR forms, complete WKT, PROJ.4, EPSG:n or a file containing the WKT.

-to <NAME>=<VALUE>

set a transformer option suitable to pass to *GDALCreateGenImgProjTransformer2()*. This is used when converting geometries coordinates to target raster pixel space. For example this can be used to specify RPC related transformer options.

-co <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--format* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

-te <xmin> <ymin> <xmax> <ymax>

Set georeferenced extents. The values must be expressed in georeferenced units. If not specified, the extent of the output file will be the extent of the vector layers.

-tr <xres> <yres>

Set target resolution. The values must be expressed in georeferenced units. Both must be positive values.

-tap

(target aligned pixels) Align the coordinates of the extent of the output file to the values of the *-tr*, such that the aligned extent includes the minimum extent. Alignment means that xmin / resx, ymin / resy, xmax / resx and ymax / resy are integer values.

-ts <width> <height>

Set output file size in pixels and lines. Note that *-ts* cannot be used with *-tr* If one of the two values is set to 0, it will be computed from the other value in order to preserve the aspect ratio.

-ot <type>

Force the output bands to be of the indicated data type. Defaults to `Float64`, unless the attribute field to burn is of type `Int64`, in which case `Int64` is used for the output raster data type if the output driver supports it.

-optim {AUTO|VECTOR|RASTER}

Force the algorithm used (results are identical). Raster mode is used in most cases and optimizes read/write operations. The vector mode is useful with a large amount of input features and optimizes CPU use, provided that the output image is tiled. Auto mode (the default) will choose the algorithm based on input and output properties.

-oo <NAME>=<VALUE>

Added in version 3.7.

Source dataset open option (format specific)

<src_datasource>

Any OGR supported readable datasource.

<dst_filename>

The GDAL supported output file. Must support update mode access. This file will be created if it does not already exist If the output raster already exists, the affected pixels are updated in-place.

The program creates a new target raster image when any of the *-of*, *-a_nodata*, *-init*, *-a_srs*, *-co*, *-te*, *-tr*, *-tap*, *-ts*, or *-ot* options are used. The resolution or size must be specified using the *-tr* or *-ts* option for all new rasters. The target raster will be overwritten if it already exists and any of these creation-related options are used.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

C API

This utility is also callable from C with *GDALRasterize()*.

Examples

Example 1

The following would burn all polygons from `mask.shp` into the RGB TIFF file `work.tif` with the color red (RGB = 255,0,0).

```
gdal_rasterize -b 1 -b 2 -b 3 -burn 255 -burn 0 -burn 0 -l mask mask.shp work.tif
```

Example 2

The following would burn all "class A" buildings into the output elevation file, pulling the top elevation from the ROOF_H attribute.

```
gdal_rasterize -a ROOF_H -where "class='A'" -l footprints footprints.shp city_dem.tif
```

Example 3

The following would burn all polygons from `footprint.shp` into a new 1000x1000 rgb TIFF as the color red. Note that *-b* is not used; the order of the *-burn* options determines the bands of the output raster.

```
gdal_rasterize -burn 255 -burn 0 -burn 0 -ot Byte -ts 1000 1000 -l footprints footprints.
↪shp mask.tif
```

5.2.2.17 gdal_retile

Synopsis

```
gdal_retile [--help] [--help-general]
               [-v] [-co <NAME>=<VALUE>]... [-of <out_format>] [-ps <pixelWidth>
↪<pixelHeight>]
               [-overlap <val_in_pixel>]
               [-ot  {Byte/Int8/Int16/UInt16/UInt32/Int32/Float32/Float64/
                      CInt16/CInt32/CFloat32/CFloat64}]'
               [ -tileIndex <tileIndexName> [-tileIndexField <tileIndexFieldName>]]
               [-csv <fileName> [-csvDelim <delimiter>]]
               [-s_srs <srs_def>]  [-pyramidOnly]
               [-r {near|bilinear|cubic|cubicspline|lanczos}]
               -levels <numberoflevels>
               [-useDirForEachRow] [-resume]
               -targetDir <TileDirectory> <input_file> <input_file>...
```

Description

This utility will retile a set of input tile(s). All the input tile(s) must be georeferenced in the same coordinate system and have a matching number of bands. The geotransform matrix of input tiles must not contain rotation terms.

Optionally pyramid levels are generated. All pyramid levels are generated from the input tiles (not from previous levels).

It is possible to generate shape file(s) for the tiled output.

If your number of input tiles exhausts the command line buffer, use the general *--optfile* option

Note

gdal_retile is a Python utility, and is only available if GDAL Python bindings are available.

Tip

Equivalent in new "gdal" command line interface:

See *gdal raster tile* (partially equivalent).

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-targetDir <directory>

The directory where the tile result is created. Pyramids are stored in sub-directories numbered from 1. Created tile names have a numbering schema and contain the name of the source tiles(s)

-of <format>

Select the output format. If not specified, the format is guessed from the extension. Use the short format name.

-co <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--format* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

-ot <type>

Force the output image bands to have a specific data type supported by the driver, which may be one of the following: `Byte`, `Int8`, `UInt16`, `Int16`, `UInt32`, `Int32`, `UInt64`, `Int64`, `Float32`, `Float64`, `CInt16`, `CInt32`, `CFloat32` or `CFloat64`.

-ps <pixelsize_x> <pixelsize_y>

Pixel size to be used for the output file. If not specified, 256 x 256 is the default

-overlap <val_in_pixel>

Overlap in pixels between consecutive tiles. If not specified, 0 is the default

-levels <numberOfLevels>

Number of pyramids levels to build.

-v

Generate verbose output of tile operations as they are done.

-pyramidOnly

No retiling, build only the pyramids

-r <algorithm>

Resampling algorithm, default is near

-s_srs <srs_def>

Source spatial reference to use. The coordinate systems that can be passed are anything supported by the OGRSpatialReference.SetFromUserInput() call, which includes EPSG, PCS, and GCSes (i.e. EPSG:4296), PROJ.4 declarations (as above), or the name of a .prj file containing well known text. If no srs_def is given, the srs_def of the source tiles is used (if there is any). The srs_def will be propagated to created tiles (if possible) and to the optional shape file(s)

-tileIndex <tileIndexName>

The name of shape file containing the result tile(s) index

-tileIndexField <tileIndexFieldName>

The name of the attribute containing the tile name

-csv <csvFileName>

The name of the csv file containing the tile(s) georeferencing information. The file contains 5 columns: tilename,minx,maxx,miny,maxy

-csvDelim <column delimiter>

The column delimiter used in the CSV file, default value is a semicolon ";"

-useDirForEachRow

Normally the tiles of the base image are stored as described in *-targetDir*. For large images, some file systems have performance problems if the number of files in a directory is too big, causing gdal_retile not to finish in reasonable time. Using this parameter creates a different output structure. The tiles of the base image are stored in a sub-directory called 0, the pyramids in sub-directories numbered 1,2,.... Within each of these directories another level of sub-directories is created, numbered from 0...n, depending of how many tile rows are needed for each level. Finally, a directory contains only the tiles for one row for a specific level. For large images a performance improvement of a factor N could be achieved.

-resume

Resume mode. Generate only missing files.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

5.2.2.18 gdal_sieve

Synopsis

```
gdal_sieve [--help] [--help-general]
              [-q] [-st threshold] [-4] [-8] [-o name=value]
              <srcfile> [-nomask] [-mask filename] [-of format] [<dstfile>]
```

Description

gdal_sieve removes raster polygons smaller than a provided threshold size (in pixels) and replaces them with the pixel value of the largest neighbour polygon. The result can be written back to the existing raster band, or copied into a new file.

The input dataset is read as integer data which means that floating point values are rounded to integers. Re-scaling source data may be necessary in some cases (e.g. 32-bit floating point data with min=0 and max=1).

Additional details on the algorithm are available in the *GDALSieveFilter()* docs.

Note

gdal_sieve is a Python utility, and is only available if GDAL Python bindings are available.

Tip

Equivalent in new "gdal" command line interface:

See *gdal raster sieve*.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

5.2.2.19 gdal_translate

Synopsis

```
Usage: gdal_translate [--help] [--long-usage] [--help-general]
                      [-ot Byte|Int8|[U]Int{16|32|64}|CInt{16|32}|[C]Float{32|64}]
                      [-if <format>]... [-of <output_format>] [--quiet]
                      [-b <band>]... [-mask <mask>] [-expand gray|rgb|rgba]
                      [[-strict]|[-not_strict]]
                      [-outsize <xsize[%]|0> <ysize[%]|0>] [-tr <xres> <yres>]
```

(continues on next page)

(continued from previous page)

```
                      [-ovr <level>|AUTO|AUTO-<n>|NONE] [-sds]
                      [-r nearest,bilinear,cubic,cubicspline,lanczos,average,mode]
                      [[-scale [<src_min> <src_max> [<dst_min> <dst_max>]]]...|
                      [-scale_X [<src_min> <src_max> [<dst_min> <dst_max>]]]...|
                      [-unscale]]
                      [[-exponent <value>]|[-exponent_X <value>]...]
                      [-srcwin <xoff> <yoff> <xsize> <ysize>]
                      [-projwin <ulx> <uly> <lrx> <lry>]
                      [-projwin_srs <srs_def>] [-epo] [-eco] [-a_srs <srs_def>]
                      [-a_coord_epoch <epoch>] [-a_ullr <ulx> <uly> <lrx> <lry>]
                      [-a_nodata <value>|none]
                      [-a_gt <gt(0)> <gt(1)> <gt(2)> <gt(3)> <gt(4)> <gt(5)>]
                      [-a_scale <value>] [-a_offset <value>] [-nogcp]
                      [-gcp <pixel> <line> <easting> <northing> [<elevation>]]...
                      [-colorinterp
↪{red|green|blue|alpha|gray|undefined|pan|coastal|rededge|nir|swir|mwir|lwir|...},...]
                      [-colorinterp_X
↪{red|green|blue|alpha|gray|undefined|pan|coastal|rededge|nir|swir|mwir|lwir|...}]...
                      [[-stats]|[-approx_stats]]
                      [-norat] [-noxmp] [-co <NAME>=<VALUE>]...
                      [-mo <NAME>=<VALUE>]... [-dmo <DOMAIN>:<KEY>=<VALUE>]...
                      [-oo <NAME>=<VALUE>]...
                      <input_file> <output_file>
```

Description

The **gdal_translate** utility can be used to convert raster data between different formats, potentially performing some operations like subsetting, resampling, and rescaling pixels in the process.

Tip

Equivalent in new "gdal" command line interface:

- *gdal raster convert* for format translation.
- *gdal raster clip* for spatial subsetting.
- *gdal raster nodata-to-alpha* to add an alpha channel from nodata values.
- *gdal raster resize* for image resizing.
- *gdal raster scale* to scale pixel values.
- *gdal raster set-type* to change the band data type.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-q, --quiet

Suppress progress messages and other non-error output.

-ot <type>

Force the output image bands to have a specific data type supported by the driver, which may be one of the following: `Byte`, `Int8`, `UInt16`, `Int16`, `UInt32`, `Int32`, `UInt64`, `Int64`, `Float32`, `Float64`, `CInt16`, `CInt32`, `CFloat32` or `CFloat64`.

-strict

Enable strict mode. In this mode, GDAL will fail instead of silently performing operations that may lead to loss of information, such as data type conversions that cannot be exactly preserved.

The exact behavior of this option is driver-dependent. Most raster drivers use it to enforce strict preservation of the input data type and will report an error if the requested operation cannot be performed without data loss. See *Example 5*.

-if <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

Added in version 3.2.

-of <format>

Select the output format. If not specified, the format is guessed from the extension. Use the short format name.

-b <band>

Select an input band **band** for output. Bands are numbered from 1. Multiple *-b* switches may be used to select a set of input bands to write to the output file, or to reorder bands. **band** can also be set to "mask,1" (or just "mask") to mean the mask band of the first band of the input dataset.

-mask <band>

Select an input band **band** to create output dataset mask band. Bands are numbered from 1. **band** can be set to "none" to avoid copying the global mask of the input dataset if it exists. Otherwise it is copied by default ("auto"), unless the mask is an alpha channel, or if it is explicitly used to be a regular band of the output dataset ("-b mask"). **band** can also be set to "mask,1" (or just "mask") to mean the mask band of the 1st band of the input dataset.

-expand gray|rgb|rgba

To expose a dataset with 1 band with a color table as a dataset with 3 (RGB) or 4 (RGBA) bands. Useful for output drivers such as JPEG, JPEG2000, MrSID, ECW that don't support color indexed datasets. The 'gray' value enables to expand a dataset with a color table that only contains gray levels to a gray indexed dataset.

-outsize <xsize>[%]|0 <ysize>[%]|0

Set the size of the output file. Outsize is in pixels and lines unless '%' is attached in which case it is as a fraction of the input image size. If one of the 2 values is set to 0, its value will be determined from the other one, while maintaining the aspect ratio of the source dataset.

-tr <xres> <yres>

Set target resolution. The values must be expressed in georeferenced units. Both must be positive values. This is mutually exclusive with *-outsize*, *-a_ullr*, and *-a_gt*.

-ovr {<level>|AUTO|AUTO-<n>|NONE}

Added in version 3.6.

To specify which overview level of source file must be used. The default choice, AUTO, will select the overview level whose resolution is the closest to the target resolution. Specify an integer value (0-based, i.e. 0=1st overview level) to select a particular level. Specify AUTO-n where n is an integer greater or equal to 1, to select an overview level below the AUTO one. Or specify NONE to force the base resolution to be used (can be useful if overviews

have been generated with a low quality resampling method, and a higher quality resampling method is specified with *-r*.)

When *-ovr* is specified as an integer value, and neither *-outsize* nor *-tr* is specified, the size of the overview will be used as the output size.

When *-ovr* is specified, values of *-srcwin* should be expressed as pixel offset and size of the full resolution source dataset. Similarly when using *-outsize* with percentage values, they refer to the size of the full resolution source dataset.

-r `{nearest|bilinear|cubic|cubicspline|lanczos|average|rms|mode}`

Select a resampling algorithm.

`nearest` (default) applies a nearest neighbour (simple sampling) resampler

`average` computes the average of all non-NODATA contributing pixels. Starting with GDAL 3.1, this is a weighted average taking into account properly the weight of source pixels not contributing fully to the target pixel.

`rms` computes the root mean squared / quadratic mean of all non-NODATA contributing pixels (GDAL >= 3.3)

`bilinear` applies a bilinear convolution kernel.

`cubic` applies a cubic convolution kernel.

`cubicspline` applies a B-Spline convolution kernel.

`lanczos` applies a Lanczos windowed sinc convolution kernel.

`mode` selects the value which appears most often of all the sampled points.

-scale `[<src_min> <src_max> [<dst_min> <dst_max>]]`

Rescale the input pixels values from the range **src_min** to **src_max** to the range **dst_min** to **dst_max**. If omitted the output range is from the minimum value to the maximum value allowed for integer data types (for example from 0 to 255 for Byte output) or from 0 to 1 for floating-point data types. If omitted the input range is automatically computed from the source dataset, in its whole (not just the window of interest potentially specified with *-srcwin* or *-projwin*). This may be a slow operation on a large source dataset, and if using it multiple times for several gdal_translate invocation, it might be beneficial to call `gdalinfo -stats {source_dataset}` priorly to precompute statistics, for formats that support serializing statistics computations (GeoTIFF, VRT...) Note that the values specified after *-scale* are only used to compute a scale and offset to apply to the input raster values. In particular, `src_min` and `src_max` are not used to clip input values unless *-exponent* is also specified. Instead of being clipped, source values that are outside the range of `src_min` and `src_max` will be scaled to values outside the range of `dst_min` and `dst_max`. If clipping without exponential scaling is desired, `-exponent 1` can be used. *-scale* can be repeated several times (if specified only once, it also applies to all bands of the output dataset), so as to specify per band parameters. It is also possible to use the `-scale_bn` syntax where bn is a band number (e.g. `-scale_2` for the 2nd band of the output dataset) to specify the parameters of one or several specific bands.

-exponent `<exp_val>`

Apply non-linear scaling with a power function. `exp_val` is the exponent of the power function (must be positive). This option must be used with the *-scale* option. If specified only once, *-exponent* applies to all bands of the output image. It can be repeated several times so as to specify per band parameters. It is also possible to use the `-exponent_bn` syntax where bn is a band number (e.g. `-exponent_2` for the 2nd band of the output dataset) to specify the parameters of one or several specific bands.

The scaled value `Dst` is calculated from the source value `Src` with the following formula:

$$Dst = (Dst_{max} - Dst_{min}) \times \max\left(0, \min\left(1, \left(\frac{Src - Src_{min}}{Src_{max} - Src_{min}}\right)^{exp_val}\right)\right) + Dst_{min}$$

-unscale

Apply the scale/offset metadata for the bands to convert linearly-scaled values to unscaled values. It is also often necessary to reset the output datatype with the *-ot* switch. The unscaled value is computed from the scaled raw value with the following formula:

$$unscaled_value = scaled_value * scale + offset$$

-srcwin <xoff> <yoff> <xsize> <ysize>

Selects a subwindow from the source image for copying based on pixel/line location. Pixel/line offsets (`xoff` and `yoff`) are measured from the left and top of the image. If the subwindow extends beyond the bounds of the source dataset, output pixels will be written with a value of zero, unless a NoData value is defined either in the source dataset or by *-a_nodata*. Alternatively, **gdal_translate** can issue an error in this case if so directed by options *-epo* or *-eco*.

-projwin <ulx> <uly> <lrx> <lry>

Selects a subwindow from the source image for copying (like *-srcwin*) but with the corners given in georeferenced coordinates (by default expressed in the SRS of the dataset. Can be changed with *-projwin_srs*). If the subwindow extends beyond the bounds of the source dataset, output pixels will be written with a value of zero, unless a NoData value is defined either in the source dataset or by *-a_nodata*. Alternatively, **gdal_translate** can issue an error in this case if so directed by options *-epo* or *-eco*.

> **Note**
>
> Beginning with GDAL 3.11, the extent described by *-projwin* will be transformed into the dataset SRS and used to select a subwindow. Before GDAL 3.11, only the two corner points were transformed into the dataset SRS, and these two transformed points were used to define an extent in the dataset SRS. Depending on the SRS involved, the subwindow selected in GDAL 3.11 may be substantially larger than in previous versions.

> **Note**
>
> When using nearest-neighbor resampling, the window specified by *-projwin* is expanded (rounded, for GDAL < 3.11) if necessary to match input pixel boundaries. For other resampling algorithms, the window is not modified.

-projwin_srs <srs_def>

Specifies the SRS in which to interpret the coordinates given with *-projwin*. The <srs_def> may be any of the usual GDAL/OGR forms, complete WKT, PROJ.4, EPSG:n or a file containing the WKT.

> **Warning**
>
> This does not cause reprojection of the dataset to the specified SRS.

-epo

(Error when Partially Outside) If this option is set, *-srcwin* or *-projwin* values that falls partially outside the source raster extent will be considered as an error. The default behavior is to accept such requests.

-eco

(Error when Completely Outside) Same as *-epo*, except that the criterion for erroring out is when the request falls completely outside the source raster extent.

-a_srs <srs_def>

Override the projection for the output file. Can be used with *-a_ullr* or *-a_gt* to specify the extent in this projection.

The coordinate reference systems that can be passed are anything supported by the *OGRSpatialReference::SetFromUserInput()* call, which includes EPSG Projected, Geographic or Compound CRS (i.e. EPSG:4296), a well known text (WKT) CRS definition, PROJ.4 declarations, or the name of a .prj file containing a WKT CRS definition.

> **Note**
>
> No reprojection is done.

-a_coord_epoch <epoch>

Added in version 3.4.

Assign a coordinate epoch, linked with the output SRS. Useful when the output SRS is a dynamic CRS.

-a_scale <value>

Set band scaling value. No modification of pixel values is done. Note that the *-unscale* does not take into account *-a_scale*. You may for example specify `-scale 0 1 <offset> <offset+scale>` to apply a (offset, scale) tuple, for the equivalent of the 2 steps: `gdal_translate input.tif tmp.vrt -a_scale scale -a_offset offset` followed by `gdal_translate tmp.vrt output.tif -unscale`

-a_offset <value>

Set band offset value. No modification of pixel values is done. Note that the *-unscale* does not take into account *-a_offset*. You may for example specify `-scale 0 1 <offset> <offset+scale>` to apply a (offset, scale) tuple, for the equivalent of the 2 steps: `gdal_translate input.tif tmp.vrt -a_scale scale -a_offset offset` followed by `gdal_translate tmp.vrt output.tif -unscale`

-a_ullr <ulx> <uly> <lrx> <lry>

Assign/override the georeferenced bounds of the output file. This assigns georeferenced bounds to the output file, ignoring what would have been derived from the source file. So this does not cause reprojection to the specified SRS. This is mutually exclusive with *-a_gt*

-a_gt <gt(0)> <gt(1)> <gt(2)> <gt(3)> <gt(4)> <gt(5)>

Assign/override the geotransform of the output file. This assigns the geotransform to the output file, ignoring what would have been derived from the source file. So this does not cause reprojection to the specified SRS. See *Geotransform Tutorial*. This is mutually exclusive with *-a_ullr*

Added in version 3.8.

-a_nodata <value>

Assign a specified nodata value to output bands. It can be set to `none` to avoid setting a nodata value to the output file if one exists for the source file. Note that, if the input dataset has a nodata value, this does not cause pixel values that are equal to that nodata value to be changed to the value specified with this option.

-colorinterp_X <red|green|blue|alpha|gray|undefined|pan|coastal|rededge|nir|swir|mwir|lwir|...>

Override the color interpretation of band X (where X is a valid band number, starting at 1)

-colorinterp {red|green|blue|alpha|gray|undefined|pan|coastal|rededge|nir|swir|mwir|lwir|...},...

Override the color interpretation of all specified bands. For example -colorinterp red,green,blue,alpha for a 4 band output dataset.

-mo <META-TAG>=<VALUE>

Passes a metadata key and value to set on the output dataset if possible.

-dmo DOMAIN:META-TAG=VALUE

Passes a metadata key and value in specified domain to set on the output dataset if possible.

Added in version 3.9.

-co <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--format* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

In addition to the driver-specific creation options, gdal_translate (and *GDALTranslate()* and *GDALCreateCopy()*) recognize the following options:

- **APPEND_SUBDATASET=[YES/NO]:** Defaults to NO.

  Can be specified to YES to avoid prior destruction of existing dataset, for drivers that support adding several subdatasets (e.g. GTIFF, NITF)

- **COPY_SRC_MDD=[AUTO/YES/NO]:** (GDAL >= 3.8) Defaults to AUTO.

  Defines if metadata domains of the source dataset should be copied to the destination dataset. In the default AUTO mode, only "safe" domains will be copied, which include the default metadata domain (some drivers may include other domains such as IMD, RPC, GEOLOCATION). When setting YES, all domains will be copied (but a few reserved ones like IMAGE_STRUCTURE or DERIVED_SUBDATASETS). Currently only recognized by the GTiff, COG, VRT, PNG and JPEG drivers.

  When setting NO, no source metadata will be copied.

- **SRC_MDD=<domain_name>:** (GDAL >= 3.8)

  Defines which source metadata domain should be copied. This option restricts the list of source metadata domains to be copied (it implies COPY_SRC_MDD=YES if it is not set). This option may be specified as many times as they are source domains. The default metadata domain is the empty string "" ("_DEFAULT_") may also be used when empty string is not practical). Currently only recognized by the GTiff, COG, VRT, PNG and JPEG drivers.

-nogcp

Do not copy the GCPs in the source dataset to the output dataset.

-gcp <pixel> <line> <easting> <northing> [<elevation>]

Add the indicated ground control point to the output dataset. This option may be provided multiple times to provide a set of GCPs.

-q

Suppress progress monitor and other non-error output.

-sds

Copy all subdatasets of this file to individual output files. Use with formats like HDF that have subdatasets.

-stats

Force (re)computation of statistics.

-norat

Do not copy source RAT into destination dataset.

-noxmp

Do not copy the XMP metadata in the source dataset to the output dataset when driver is able to copy it.

Added in version 3.2.

-oo <NAME>=<VALUE>

Dataset open option (format specific)

<src_dataset>

The source dataset name. It can be either file name, URL of data source or subdataset name for multi-dataset files.

<dst_dataset>

The destination file name.

Nodata / source validity mask handling during resampling

Masked values, either identified through a nodata value metadata set on the source band, a mask band, an alpha band will not be used during resampling (when using *-outsize* or *-tr*).

orphan

The details of how it is taken into account depends on the resampling kernel:

- for nearest resampling, for each target pixel, one of the potential contributing source pixels is selected (in an implementation specific way). Its value is used as it, be it valid or invalid.
- for bilinear, cubic, cubicspline and lanczos, for each target pixel, the weights of contributing source pixels is set to zero to ignore them when they are masked. There is an extra specificity for cubic: given that some of the weights in the kernel are negative, such strategy could lead to strong overshoot/undershoot when there is an alternance of valid and invalid pixels. Consequently, if any of the horizontal or vertical direction, if the maximum number of valid source pixels in each dimension is less than the radius of the resampling kernel, the target pixel is considered as nodata.
- for the other resampling methods, source pixels contributing to the target pixel are ignored if masked. Only the valid ones are taken into account. If there are none, the target pixel is considered as nodata.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

C API

This utility is also callable from C with *GDALTranslate()*.

Examples

Example 1: Creating a tiled GeoTIFF

```
gdal_translate -of GTiff -co "TILED=YES" utm.tif utm_tiled.tif
```

Example 2: Creating a JPEG-compressed TIFF with internal mask from a RGBA dataset

```
gdal_translate rgba.tif withmask.tif -b 1 -b 2 -b 3 -mask 4 -co COMPRESS=JPEG \
  -co PHOTOMETRIC=YCBCR --config GDAL_TIFF_INTERNAL_MASK YES
```

Example 3: Creating a RGBA dataset from a RGB dataset with a mask

```
gdal_translate withmask.tif rgba.tif -b 1 -b 2 -b 3 -b mask
```

Example 4: Subsetting using -projwin and -outsize

```
gdal_translate -projwin -20037500 10037500 0 0 -outsize 100 100 frmt_wms_googlemaps_tms.
↪xml junk.png
```

Example 5: Use of strict mode with unsupported data type

```
$ gdal_create test.tif -bands 3 -ot Int16 -outsize 1 1
$ gdal_translate -strict test.tif test.webp
ERROR 6: WEBP driver doesn't support data type Int16.
Only UInt8 bands supported.
```

5.2.2.20 gdal_viewshed

Synopsis

```
Usage: gdal_viewshed [--help] [--long-usage] [--help-general]
                     [-of <output_format>] [-ox <value>] [-oy <value>]
                     [-oz <value>] [-vv <value>] [-iv <value>] [-ov <value>]
                     [-co <NAME>=<VALUE>]... [-a_nodata <value>] [-tz <value>]
                     [-md <value>] [-j <value>] [-cc <value>] [-b <value>]
                     [-om NORMAL|DEM|GROUND|ACCUM] [-os <value>] [-sd <value>]
                     [--quiet]
                     <src_filename> <dst_filename>
```

Description

By default the **gdal_viewshed** generates a binary visibility raster from one band of the input raster elevation model (DEM). The output raster will be of type Byte. With the -mode flag can also return a minimum visible height raster of type Float64.

> **Note**
>
> The algorithm as implemented currently will only output meaningful results if the georeferencing is in a projected coordinate reference system.

> **Tip**
>
> Equivalent in new "gdal" command line interface:

See *gdal raster viewshed*

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-q, --quiet

Suppress progress messages and other non-error output.

-co <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--format* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

-b <band>

Select an input band **band** containing the DEM data. Bands are numbered from 1. Only a single band can be used. Only the part of the raster within the specified maximum distance around the observer point is processed.

-a_nodata <value>

The value to be set for the cells in the output raster that have no data.

> **Note**
>
> Currently, no special processing of input cells at a nodata value is done (which may result in erroneous results).

-ox <value>

The X position of the observer (in SRS units). If the coordinate is outside of the raster, all space between the observer and the raster is assumed not to occlude visibility of the raster. (Not supported in cumulative mode.)

-oy <value>

The Y position of the observer (in SRS units). If the coordinate is outside of the raster, all space between the observer and the raster is assumed not to occlude visibility of the raster. (Not supported in cumulative mode.)

-oz <value>

The height of the observer above the DEM surface in the height unit of the DEM. Default: 2

-tz <value>

The height of the target above the DEM surface in the height unit of the DEM. Default: 0

-md <value>

Maximum distance from observer to compute visibility. It is also used to clamp the extent of the output raster. (Not supported in cumulative mode)

-cc <value>

Coefficient to consider the effect of the curvature and refraction. When calculating visibility between two points (i.e. Line Of Sight or Viewshed), The magnitude of this effect varies with atmospheric conditions and depends on the wavelength.

Different applications for calculating visibility use different interchangeable notation to describe this phenomena: Refraction Coefficient, Curvature Coefficient, and Sphere Diameter Factor. gdal_viewshed uses the Curvature Coefficient notation.

$$CurvCoeff = 1 - RefractionCoeff$$

Changes in air density curve the light downward causing an observer to see further and the earth to appear less curved, as if the sphere (earth) diameter is larger then it actually is. The ratio between that imaginary sphere diameter and the actual sphere diameter is given by the formula:

$$SphereDiameterFactor = 1/CurvCoeff = 1/(1 - RefractionCoeff)$$

For visible light, the standard atmospheric refraction coefficient that is generally used is 1/7. Thus the default value (since GDAL 3.4) for CurvCoeff that gdal_viewshed uses is 0.85714 (=~ 1-1/7) for Earth CRS. Starting with GDAL 3.6, for non-Earth CRS (those whole semi-major axis differs by more than 5% with the one of WGS 84), CurvCoeff default value is 1.0, to account for the no refraction use case.

The height of the DEM is corrected according to the following formula:

$$Height_{Corrected} = Height_{DEM} - CurvCoeff \frac{TargetDistance^2}{SphereDiameter}$$

Typical coefficient values are given in the table below (use Curvature Coeff value for the cc option)

Use Case	Refraction Coeff	**Curvature Coeff**	Sphere Diameter Factor
No Refraction	0	1	1
Visible Light	1/7	6/7 (=~0.85714)	7/6 (=~1.1666)
Radio Waves	0.25 ~ 0.325	0.75 ~ 0.675	1.33 ~ 1.48
Flat Earth	1	0	inf

-iv <value>

Pixel value to set for invisible areas. (Not supported in cumulative mode) Default: 0

-ov <value>

Pixel value to set for the cells that fall outside of the range specified by the observer location and the maximum distance. (Not supported in cumulative mode) Default: 0

-vv <value>

Pixel value to set for visible areas. (Not supported in cumulative mode) Default: 255

-om <output mode>

Sets what information the output contains.

Possible values: NORMAL, DEM, GROUND, ACCUM

NORMAL returns a raster of type Byte containing visible locations.

DEM and GROUND will return a raster of type Float64 containing the minimum target height for target to be visible from the DEM surface or ground level respectively. That is to say, if the minimum target height for the target to be visible at a point is `h` and the value of the input raster at that point is `E`, for `DEM`, `E + h` will be the output value. For `ground`, `h` will be output value. Flags -tz, -iv and -vv will be ignored.

Cumulative (ACCUM) mode will create an eight bit raster the same size as the input raster where each cell represents the relative observability from a grid of observer points. See the -os option.

Default NORMAL

-os <value>

Cell Spacing between observers (only supported in cumulative mode) Default: 10

-j <value>

Number of jobs to run at once. (only supported in cumulative mode) Default: 3

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

C API

Functionality of this utility can be done from C with *GDALViewshedGenerate()*.

Examples

Example 1

Screenshot of 2 combined viewshed analysis, with the yellow pixels showing the area that is visible from the both observation locations (the green dots), while the small green area is only visible from one location.

Create a viewshed raster with a radius of 500 for a person standing at location (-10147017, 5108065).

```
gdal_viewshed -md 500 -ox -10147017 -oy 5108065 source.tif destination.tif
```

Reference

5.2.2.21 gdaladdo

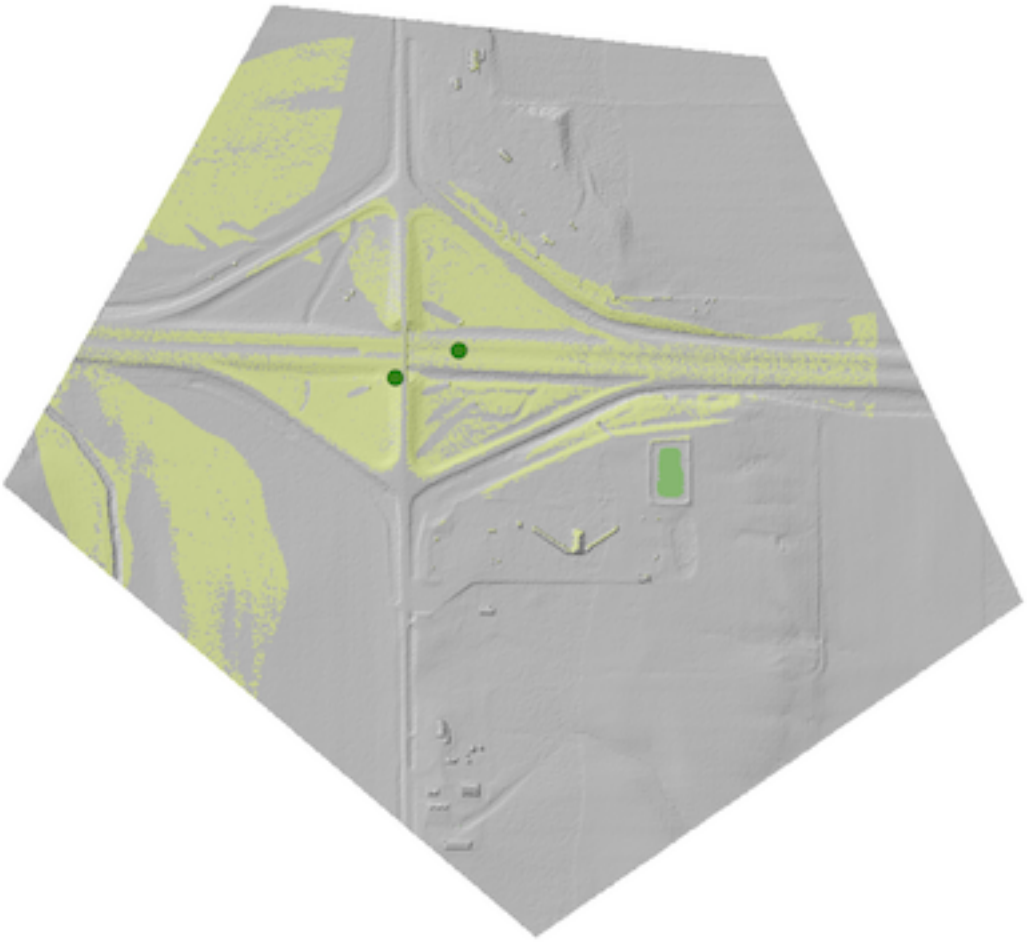
Synopsis

```
Usage: gdaladdo [--help] [--long-usage] [--help-general]
                [-r nearest|average|rms|gauss|bilinear|cubic|cubicspline|lanczos|average_
↪magphase|mode]
                [-ro] [--quiet] [-b <band>]... [-oo <NAME=VALUE>]...
                [-minsize <val>]
                [[-clean]|[--partial-refresh-from-source-timestamp]|
                [--partial-refresh-from-projwin <ulx> <uly> <lrx> <lry>]|
                [--partial-refresh-from-source-extent <filename1>[,<filenameN>]...]]
                filename [<level>]...
```

Description

The **gdaladdo** utility can be used to build or rebuild overview images for most supported file formats with one of several downsampling algorithms.

> **Tip**
>
> Equivalent in new "gdal" command line interface:
>
> See *gdal raster overview add*, *gdal raster overview delete* and *gdal raster overview refresh*.



--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-q, --quiet

Suppress progress messages and other non-error output.

-r {nearest|average|rms|gauss|bilinear|cubic|cubicspline|lanczos|average_magphase|mode}

Select a resampling algorithm. The default is `nearest`, which is generally not appropriate if sub-pixel accuracy is desired.

Starting with GDAL 3.9, when refreshing existing TIFF overviews, the previously used method, as noted in the RESAMPLING metadata item of the overview, will be used if *-r* is not specified.

The available methods are:

`nearest` applies a nearest neighbour (simple sampling) resampler.

`average` computes the average of all non-NODATA contributing pixels. Starting with GDAL 3.1, this is a weighted average taking into account properly the weight of source pixels not contributing fully to the target pixel.

`rms` computes the root mean squared / quadratic mean of all non-NODATA contributing pixels (GDAL >= 3.3)

`gauss` applies a Gaussian kernel before computing the overview, which can lead to better results than simple averaging in e.g case of sharp edges with high contrast or noisy patterns. The advised level values should be 2, 4, 8, ... so that a 3x3 resampling Gaussian kernel is selected.

`bilinear` applies a bilinear convolution kernel.

`cubic` applies a cubic convolution kernel.

`cubicspline` applies a B-Spline convolution kernel.

`lanczos` applies a Lanczos windowed sinc convolution kernel.

`average_magphase` averages complex data in mag/phase space.

`mode` selects the value which appears most often of all the sampled points.

-b <band>

Select an input band **band** for overview generation. Band numbering starts from 1. Multiple *-b* switches may be used to select a set of input bands to generate overviews.

-ro

open the dataset in read-only mode, in order to generate external overview (for GeoTIFF especially).

-clean

remove all overviews.

-oo <NAME>=<VALUE>

Dataset open option (format specific)

-minsize <val>

Maximum width or height of the smallest overview level. Only taken into account if explicit levels are not specified. Defaults to 256.

--partial-refresh-from-source-timestamp

Added in version 3.8.

This option performs a partial refresh of existing overviews, when <filename> is a VRT file with an external overview. It checks the modification timestamp of all the sources of the VRT and regenerate the overview for areas corresponding to sources whose timestamp is more recent than the external overview of the VRT. By default all existing overview levels will be refreshed, unless explicit levels are specified. See *Example 7*.

--partial-refresh-from-projwin `<ulx> <uly> <lrx> <lry>`

Added in version 3.8.

This option performs a partial refresh of existing overviews, in the region of interest specified by georeference coordinates where <ulx> is the X value of the upper left corner, <uly> is the Y value of the upper left corner, <lrx> is the X value of the lower right corner and <lry> is the Y value of the lower right corner. By default all existing overview levels will be refreshed, unless explicit levels are specified.

--partial-refresh-from-source-extent `<filename1>[,<filenameN>]...`

Added in version 3.8.

This option performs a partial refresh of existing overviews, in the region of interest specified by one or several filenames (names separated by comma). Note that the filenames are only used to determine the regions of interest to refresh. The reference source pixels are the one of the main dataset. By default all existing overview levels will be refreshed, unless explicit levels are specified. See *Example 8*.

<filename>

The file to build overviews for (or whose overviews must be removed).

<levels>

A list of integral overview levels to build. Ignored with `-clean` option.

> Levels are no longer required to build overviews. In which case, appropriate overview power-of-two factors will be selected until the smallest overview is smaller than the value of the -minsize switch.
>
> Starting with GDAL 3.9, if there are already existing overviews, the corresponding levels will be used to refresh them if no explicit levels are specified.

gdaladdo will honour properly NODATA_VALUES tuples (special dataset metadata) so that only a given RGB triplet (in case of a RGB image) will be considered as the nodata value and not each value of the triplet independently per band.

Selecting a level value like 2 causes an overview level that is 1/2 the resolution (in each dimension) of the base layer to be computed. If the file has existing overview levels at a level selected, those levels will be recomputed and rewritten in place.

For internal GeoTIFF overviews (or external overviews in GeoTIFF format), note that -clean does not shrink the file. A later run of gdaladdo with overview levels will cause the file to be expanded, rather than reusing the space of the previously deleted overviews. If you just want to change the resampling method on a file that already has overviews computed, you don't need to clean the existing overviews.

Some format drivers do not support overviews at all. Many format drivers store overviews in a secondary file with the extension .ovr that is actually in TIFF format. By default, the GeoTIFF driver stores overviews internally to the file operated on (if it is writable), unless the -ro flag is specified.

Most drivers also support an alternate overview format using Erdas Imagine format. To trigger this use the *USE_RRD=YES* configuration option (*Example 5*). This will place the overviews in an associated .aux file suitable for direct use with Imagine or ArcGIS as well as GDAL applications. (e.g. --config USE_RRD YES)

External overviews in GeoTIFF format

External overviews created in TIFF format may be compressed using the *COMPRESS_OVERVIEW* configuration option. All compression methods, supported by the GeoTIFF driver, are available here. (e.g. `--config COMPRESS_OVERVIEW DEFLATE`). The photometric interpretation can be set with the *PHOTOMETRIC_OVERVIEW* =RGB/YCBCR/... configuration option, and the interleaving with the *INTERLEAVE_OVERVIEW* =PIXEL/BAND configuration option.

Since GDAL 3.6, *COMPRESS_OVERVIEW* and *INTERLEAVE_OVERVIEW* are honoured when creating internal overviews of TIFF files.

For JPEG compressed external and internal overviews, the JPEG quality can be set with `--config JPEG_QUALITY_OVERVIEW value`.

For WEBP compressed external and internal overviews, the WEBP quality level can be set with `--config WEBP_LEVEL_OVERVIEW value`. If not set, will default to 75.

For WEBP compressed external and internal overviews, the WEBP lossless/lossy switch can be set with `--config WEBP_LOSSLESS_OVERVIEW value`. If not set, will default to NO (lossy). Added in GDAL 3.6.0

For LERC compressed external and internal overviews, the max error threshold can be set with `--config MAX_Z_ERROR_OVERVIEW value`. If not set, will default to 0 (lossless). Added in GDAL 3.4.1

For DEFLATE or LERC_DEFLATE compressed external and internal overviews, the compression level can be set with `--config ZLEVEL_OVERVIEW value`. If not set, will default to 6. Added in GDAL 3.4.1

For ZSTD or LERC_ZSTD compressed external and internal overviews, the compression level can be set with `--config ZSTD_LEVEL_OVERVIEW value`. If not set, will default to 9. Added in GDAL 3.4.1

For JPEG-XL compressed external and internal overviews, the following settings can be set since GDAL 3.9.0:

- Whether compression should be lossless with `--config JXL_LOSSLESS_OVERVIEW YES|NO`. Default is YES
- Level of effort with `--config JXL_EFFORT_OVERVIEW value`, with value between 1(fast) and 9(flow). Default is 5
- Distance level for lossy compression with `--config JXL_DISTANCE_OVERVIEW value`, with value: 0=mathematically lossless, 1.0=visually lossless, usual range [0.5,3]. Default is 1.0. Ignored if JXL_LOSSLESS_OVERVIEW is YES
- Distance level for lossy compression of alpha channel with `--config JXL_ALPHA_DISTANCE_OVERVIEW value`, with value: 0=mathematically lossless, 1.0=visually lossless, usual range [0.5,3]. Default is the same value as JXL_DISTANCE_OVERVIEW. Ignored if JXL_LOSSLESS_OVERVIEW is YES

For LZW, ZSTD or DEFLATE compressed external overviews, the predictor value can be set with `--config PREDICTOR_OVERVIEW 1|2|3`.

To produce the smallest possible JPEG-In-TIFF overviews, you should use:

```
--config COMPRESS_OVERVIEW JPEG --config PHOTOMETRIC_OVERVIEW YCBCR --config INTERLEAVE_
↪OVERVIEW PIXEL
```

External overviews can be created in the BigTIFF format by using the *BIGTIFF_OVERVIEW* configuration option: `--config BIGTIFF_OVERVIEW {IF_NEEDED|IF_SAFER|YES|NO}`.

The default value is IF_SAFER. The behavior of this option is exactly the same as the BIGTIFF creation option documented in the GeoTIFF driver documentation.

- YES forces BigTIFF.
- NO forces classic TIFF.
- IF_NEEDED will only create a BigTIFF if it is clearly needed (uncompressed, and overviews larger than 4GB).
- IF_SAFER will create BigTIFF if the resulting file *might* exceed 4GB.

Sparse GeoTIFF overview files (that is tiles which are omitted if all their pixels are at the nodata value, when there's one, or at 0 otherwise) can be obtained with `--config SPARSE_OK_OVERVIEW ON`. Added in GDAL 3.4.1

See the documentation of the *GTiff -- GeoTIFF File Format* driver for further explanations on all those options.

Setting blocksize in Geotiff overviews

`--config GDAL_TIFF_OVR_BLOCKSIZE <size>`

Example: `--config GDAL_TIFF_OVR_BLOCKSIZE 256`

Default value is 128, or starting with GDAL 3.1, if creating overviews on a tiled GeoTIFF file, the tile size of the full resolution image. Note: without this setting, the file can have the full resolution image with a blocksize different from overviews blocksize.(e.g. full resolution image at blocksize 256, overviews at blocksize 128)

Nodata / source validity mask handling during resampling

Invalid values in source pixels, either identified through a nodata value metadata set on the source band, a mask band, an alpha band will not be used during resampling.

orphan

The details of how it is taken into account depends on the resampling kernel:

- for nearest resampling, for each target pixel, one of the potential contributing source pixels is selected (in an implementation specific way). Its value is used as it, be it valid or invalid.
- for bilinear, cubic, cubicspline and lanczos, for each target pixel, the weights of contributing source pixels is set to zero to ignore them when they are masked. There is an extra specificity for cubic: given that some of the weights in the kernel are negative, such strategy could lead to strong overshoot/undershoot when there is an alternance of valid and invalid pixels. Consequently, if any of the horizontal or vertical direction, if the maximum number of valid source pixels in each dimension is less than the radius of the resampling kernel, the target pixel is considered as nodata.
- for the other resampling methods, source pixels contributing to the target pixel are ignored if masked. Only the valid ones are taken into account. If there are none, the target pixel is considered as nodata.

Multithreading

Added in version 3.2.

The *`GDAL_NUM_THREADS`* configuration option can be set to `ALL_CPUS` or a integer value to specify the number of threads to use for overview computation.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

C API

Functionality of this utility can be done from C with *`GDALBuildOverviews()`*.

Examples

Example 1: Create overviews, embedded in the supplied TIFF file, with automatic computation of levels

```
gdaladdo -r average abc.tif
```

Example 2: Create overviews, embedded in the supplied TIFF file

```
gdaladdo -r average abc.tif 2 4 8 16
```

Example 3: Create an external compressed GeoTIFF overview file from the ERDAS .IMG file

```
gdaladdo -ro --config COMPRESS_OVERVIEW=YES erdas.img 2 4 8 16
```

Example 4: Create an external JPEG-compressed GeoTIFF overview file from a 3-band RGB dataset

If the dataset is a writable GeoTIFF, you also need to add the *-ro* option to force the generation of external overview.

```
gdaladdo --config COMPRESS_OVERVIEW=JPEG --config PHOTOMETRIC_OVERVIEW=YCBCR
         --config INTERLEAVE_OVERVIEW=PIXEL rgb_dataset.ext 2 4 8 16
```

Example 5: Create Erdas Imagine format overviews for the indicated JPEG file

```
gdaladdo --config USE_RRD=YES airphoto.jpg 3 9 27 81
```

Example 6: Create overviews for a specific subdataset

For example, one of potentially many raster layers in a GeoPackage (the "filename" parameter must be driver prefix, filename and subdataset name, like e.g. shown by gdalinfo):

```
gdaladdo GPKG:file.gpkg:layer
```

Example 7: Refresh overviews of a VRT file

This is needed when for sources have been modified after the .vrt.ovr generation:

```
gdalbuildvrt my.vrt tile1.tif tile2.tif                        # create VRT
gdaladdo -r cubic my.vrt                                       # initial overview␣
↪generation
touch tile1.tif                                                # simulate update of␣
↪one of the source tiles
gdaladdo --partial-refresh-from-source-timestamp -r cubic my.vrt # refresh overviews
```

Example 8: Refresh overviews of a TIFF file

```
gdalwarp -overwrite tile1.tif tile2.tif mosaic.tif                        # create␣
↪mosaic
gdaladdo -r cubic mosaic.tif                                              # initial␣
↪overview generation
touch tile1.tif                                                           # simulate␣
```

(continues on next page)

(continued from previous page)

```
↪update of one of the source tiles
gdalwarp tile1.tif mosaic.tif                                          # update␣
↪mosaic
gdaladdo --partial-refresh-from-source-extent tile1.tif -r cubic mosaic.tif # refresh␣
↪overviews
```

5.2.2.22 gdalattachpct

Synopsis

```
gdalattachpct [--help] [--help-general]
                 [-of format] <palette_file> <source_file> <dest_file>
```

Description

This utility will attach a color table file from an input raster file or a color table file to another raster.

> **Note**
>
> gdalattachpct is a Python utility, and is only available if GDAL Python bindings are available.

--help
: Show this help message and exit

--help-general
: Gives a brief usage message for the generic GDAL commandline options and exit.

-of <format>
: Select the output format. If not specified, the format is guessed from the extension. Use the short format name.

<palette_file>
: Extract the color table from <palette_file>. The<palette_file> must be either a raster file in a GDAL supported format with a palette or a color file in a supported format (txt, qml, qlr).

<source_file>
: The input file.

<dest_file>
: The output RGB file that will be created.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

5.2.2.23 gdalbuildvrt

Synopsis

```
gdalbuildvrt [--help] [--long-usage] [--help-general]
             [--quiet]
```

(continues on next page)

(continued from previous page)

```
            [[-strict]|[-non_strict]]
            [-tile_index <field_name>]
            [-resolution user|average|common|highest|lowest|same]
            [-tr <xres> <yres>] [-input_file_list <filename>]
            [[-separate]|[-pixel-function <function>]]
            [-pixel-function-arg <NAME>=<VALUE>]...
            [-allow_projection_difference] [-sd <n>] [-tap]
            [-te <xmin> <ymin> <xmax> <ymax>] [-addalpha] [-b <band>]...
            [-hidenodata] [-overwrite]
            [-srcnodata "<value>[ <value>]..."]
            [-vrtnodata "<value>[ <value>]..."] [-a_srs <srs_def>]
            [-r nearest|bilinear|cubic|cubicspline|lanczos|average|mode]
            [-oo <NAME>=<VALUE>]... [-co <NAME>=<VALUE>]...
            [-ignore_srcmaskband]
            [-nodata_max_mask_threshold <threshold>]
            <vrt_dataset_name> [<src_dataset_name>]...
```

Description

This program builds a raster *VRT (Virtual Dataset)* that is a mosaic of a list of input GDAL datasets. The list of input GDAL datasets can be specified at the end of the command line, put in a text file (one filename per line) for very long lists, or it can be a MapServer tileindex (see the *gdaltindex* utility). If using a tile index, all entries in the tile index will be added to the VRT.

> **Note**
>
> Starting with GDAL 3.9, for virtual mosaics with a very large number of source rasters (hundreds of thousands of source rasters, or more), it is advised to use the *gdaltindex* utility to generate a tile index compatible with the *GTI* driver.

> **Tip**
>
> Equivalent in new "gdal" command line interface:
>
> See *gdal raster mosaic* and *gdal raster stack*.

With `-separate`, each input goes into a separate band in the VRT dataset. Otherwise, the files are considered as source rasters of a larger mosaic and the VRT file has the same number of bands as the input files.

If one GDAL dataset is made of several subdatasets and has 0 raster bands, all the subdatasets will be added to the VRT rather than the dataset itself.

gdalbuildvrt does some checks to ensure that all files that will be put in the resulting VRT have similar characteristics: number of bands, projection, color interpretation, etc. If not, files that do not match the common characteristics will be skipped unless `-strict` is used. (This is only true in the default mode, and not when using the `-separate` option)

Starting with GDAL 3.12, a function (e.g., `min`, `mean`, `median`) can be specified (`-pixel-function`) to calculate pixel values from overlapping inputs. If no function is specified, or in earlier versions, the order of the input list is used to determine priority. Files that are listed at the end are the ones from which the content will be fetched. Note that nodata will be taken into account to potentially fetch data from lower-priority datasets, but currently, alpha channel

is not taken into account to do alpha compositing (so a source with alpha=0 appearing on top of another source will override its content). This might be changed in later versions.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-q, --quiet

Suppress progress messages and other non-error output.

-tileindex <field_name>

Use the specified value as the tile index field, instead of the default value which is 'location'.

-resolution {highest|lowest|average|user|same}

In case the resolution of all input files is not the same, the *-resolution* flag enables the user to control the way the output resolution is computed.

highest will pick the smallest values of pixel dimensions within the set of source rasters.

lowest will pick the largest values of pixel dimensions within the set of source rasters.

average is the default and will compute an average of pixel dimensions within the set of source rasters.

user must be used in combination with the *-tr* option to specify the target resolution.

same (added in GDAL 3.11) checks that all source rasters have the same resolution and errors out when this is not the case.

common (added in GDAL 3.11) determines the greatest common divisor of the source pixel dimensions, e.g. 0.2 for source pixel dimensions of 0.4 and 0.6.

-tr <xres> <yres>

Set target resolution. The values must be expressed in georeferenced units. Both must be positive values. Specifying these values is of course incompatible with highest|lowest|average values for *-resolution* option.

-tap

(target aligned pixels) align the coordinates of the extent of the output file to the values of the *-tr*, such that the aligned extent includes the minimum extent. Alignment means that xmin / resx, ymin / resy, xmax / resx and ymax / resy are integer values.

-te <xmin> <ymin> <xmax> <ymax>

Set georeferenced extents of VRT file. The values must be expressed in georeferenced units. If not specified, the extent of the VRT is the minimum bounding box of the set of source rasters. Pixels within the extent of the VRT but not covered by a source raster will be read as valid pixels with a value of zero unless a NODATA value is specified using *-vrtnodata* or an alpha mask band is added with *-addalpha*.

-addalpha

Adds an alpha mask band to the VRT when the source raster have none. Mainly useful for RGB sources (or grey-level sources). The alpha band is filled on-the-fly with the value 0 in areas without any source raster, and with value 255 in areas with source raster. The effect is that a RGBA viewer will render the areas without source rasters as transparent and areas with source rasters as opaque. This option is not compatible with *-separate*.

-hidenodata

Even if any band contains nodata value, giving this option makes the VRT band not report the NoData. Useful when you want to control the background color of the dataset. By using along with the -addalpha option, you can prepare a dataset which doesn't report nodata value but is transparent in areas with no data.

-srcnodata "<value>[<value>]..."

Set nodata values for input bands (different values can be supplied for each band). If more than one value is supplied all values should be quoted to keep them together as a single operating system argument. If the option is not specified, the intrinsic nodata settings on the source datasets will be used (if they exist). The value set by this option is written in the NODATA element of each `ComplexSource` element. Use a value of `None` to ignore intrinsic nodata settings on the source datasets.

-ignore_srcmaskband

Added in version 3.3.

Starting with GDAL 3.3, if a source has a mask band (internal/external mask band, or alpha band), a `<ComplexSource>` element is created by default with a `<UseMaskBand>true</UseMaskBand>` child element, to instruct the VRT driver to use the mask band of the source to mask pixels being composited. This is a generalization of the NODATA element. When specifying the *-ignore_srcmaskband* option, the mask band of sources will not be taken into account, and in case of overlapping between sources, the last one will override previous ones in areas of overlap.

-nodata_max_mask_threshold <threshold>

Added in version 3.9.

Insert a `<NoDataFromMaskSource>` source, which replaces the value of the source with the value of *-vrtnodata* (or 0 if not specified) when the value of the mask band of the source is less or equal to the threshold. This is typically used to transform a R,G,B,A image into a R,G,B one with a NoData value.

-b <band>

Select an input <band> to be processed. Bands are numbered from 1. If input bands not set all bands will be added to the VRT. Multiple *-b* switches may be used to select a set of input bands.

-sd <n>

If the input dataset contains several subdatasets, use a subdataset with the specified number (starting from 1). This is an alternative to giving the full subdataset name as an input to the utility.

-vrtnodata "<value>[<value>]..."

Set nodata values at the VRT band level (different values can be supplied for each band). If more than one value is supplied, all values should be quoted to keep them together as a single operating system argument (*Example 4*). If the option is not specified, intrinsic nodata settings on the first dataset will be used (if they exist). The value set by this option is written in the `NoDataValue` element of each `VRTRasterBand element`. Use a value of *None* to ignore intrinsic nodata settings on the source datasets.

-separate

Place each input file into a separate band. See *Example 3*. Contrary to the default mode, it is not required that all bands have the same datatype. This option is mutually exclusive with *-pixel-function*.

Starting with GDAL 3.8, all bands of each input file are added as separate VRT bands, unless *-b* is specified to select a subset of them. Before GDAL 3.8, only the first band of each input file was placed into a new VRT band, and *-b* was ignored.

-pixel-function

Specify a function name to calculate a value from overlapping inputs. For a list of available pixel functions, see *Built-in Pixel Functions*. If no function is specified, values will be taken from the last overlapping input. This option is mutually exclusive with with *-separate*.

Added in version 3.12.

-pixel-function-arg

Specify an argument to be provided to a pixel function, in the format `<NAME>=<VALUE>`. Multiple arguments may be specified by repeating this option.

Added in version 3.12.

-allow_projection_difference

When this option is specified, the utility will create a VRT even if the input datasets do not have the same projection. Note: this does not mean that they will be reprojected. Their projection will just be ignored.

-a_srs <srs_def>

Override the projection for the output file. The <srs_def> may be any of the usual GDAL/OGR forms, complete WKT, PROJ.4, EPSG:n or a file containing the WKT. No reprojection is done.

-r {nearest|bilinear|cubic|cubicspline|lanczos|average|mode}

Select a resampling algorithm. Nearest is the default

-oo <NAME>=<VALUE>

Dataset open option (format-specific)

-co <NAME>=<VALUE>

Specify a *VRT driver creation option*.

Added in version 3.10.

-input_file_list <filename>

Specify a text file with an input filename on each line. See *Example 2*.

-q

Disable the progress bar on the console

-overwrite

Overwrite the VRT if it already exists.

-strict

Turn warnings as failures. This is mutually exclusive with *-non_strict*, the latter which is the default.

Added in version 3.4.2.

-non_strict

Skip source datasets that have issues with warnings, and continue processing. This is the default.

Added in version 3.4.2.

-write_absolute_path

Added in version 3.12.0.

Enables writing the absolute path of the input datasets. By default, input filenames are written in a relative way with respect to the VRT filename (when possible).

Examples

Example 1: Make a virtual mosaic from all TIFF files contained in a directory

```
gdalbuildvrt doq_index.vrt doq/*.tif
```

Example 2: Make a virtual mosaic from files whose name is specified in a text file

```
gdalbuildvrt -input_file_list my_list.txt doq_index.vrt
```

Example 3: Make a RGB virtual mosaic from 3 single-band input files

```
gdalbuildvrt -separate rgb.vrt red.tif green.tif blue.tif
```

Example 4: Make a virtual mosaic with blue background colour (RGB: 0 0 255)

```
gdalbuildvrt -hidenodata -vrtnodata "0 0 255" doq_index.vrt doq/*.tif
```

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

C API

This utility is also callable from C with *GDALBuildVRT()*.

5.2.2.24 gdalcompare

Synopsis

```
gdalcompare [--help] [--help-general]
               [-dumpdiffs] [-skip_binary] [-skip_overviews]
               [-skip_geolocation] [-skip_geotransform]
               [-skip_metadata] [-skip_rpc] [-skip_srs]
               [-sds] <golden_file> <new_file>
```

Description

gdalcompare compares two GDAL supported datasets and reports the differences. In addition to reporting differences to the standard output the script will also return the difference count in its exit value.

Image pixels, and various metadata are checked. There is also a byte by byte comparison done which will count as one difference. So if it is only important that the GDAL visible data is identical a difference count of 1 (the binary difference) should be considered acceptable.

> **Note**
>
> gdalcompare is a Python utility, and is only available if GDAL Python bindings are available.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-dumpdiffs

Added in version 3.8.

Whether to output the difference in pixel content in a TIFF file in the current directory.

-skip_binary

Added in version 3.8.

Whether to skip exact comparison of binary content.

-skip_overviews

Added in version 3.8.

Whether to skip comparison of overviews.

-skip_geolocation

Added in version 3.8.

Whether to skip comparison of GEOLOCATION metadata domain.

-skip_geotransform

Added in version 3.8.

Whether to skip comparison of geotransform matrix.

-skip_metadata

Added in version 3.8.

Whether to skip comparison of metadata

-skip_rpc

Added in version 3.8.

Whether to skip comparison of Rational Polynomial Coefficients (RPC) metadata domain.

-skip_srs

Added in version 3.8.

Whether to skip comparison of spatial reference systems (SRS).

-sds

If this flag is passed the script will compare all subdatasets that are part of the dataset, otherwise subdatasets are ignored.

<golden_file>

The file that is considered correct, referred to as the golden file.

<new_file>

The file being compared to the golden file, referred to as the new file.

Note that the **gdalcompare** script (like all the other scripts) can also be called as a library from python code: *from osgeo_utils import gdalcompare*. The primary entry point is *gdalcompare.compare_db()* which takes a golden *gdal.Dataset* and a new *gdal.Dataset* as arguments and returns a difference count (excluding the binary comparison). The *gdalcompare.compare_sds()* entry point can be used to compare subdatasets.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

```
gdalcompare -dumpdiffs N.tiff S.tiff; echo $?
Files differ at the binary level.
Band 1 checksum difference:
  Golden: 36694
  New:    40645
  Pixels Differing: 1509
  Maximum Pixel Difference: 255.0
  Wrote Diffs to: 1.tif
Differences Found: 2
2

gdalcompare N.tiff N.tiff; echo $?
Differences Found: 0
0
```

5.2.2.25 gdaldem

Synopsis

```
gdaldem [--help] [--help-general] <mode> <input> <output> <options>
```

From any GDAL-supported elevation raster:

Generate a shaded relief map:

```
gdaldem hillshade <input_dem> <output_hillshade>
            [-z <zfactor>] [[-s <scale>] | [-xscale <xscale> -yscale <yscale>]]
            [-az <azimuth>] [-alt <altitude>]
            [-alg ZevenbergenThorne] [-combined | -multidirectional | -igor]
            [-compute_edges] [-b <Band>] [-of <format>] [-co <NAME>=<VALUE>]... [-q]
```

Generate a slope map:

```
gdaldem slope <input_dem> <output_slope_map>
            [-p] [[-s <scale>] | [-xscale <xscale> -yscale <yscale>]]
            [-alg ZevenbergenThorne]
            [-compute_edges] [-b <band>] [-of <format>] [-co <NAME>=<VALUE>]... [-q]
```

Generate an aspect map, outputs a 32-bit float raster with pixel values from 0-360 indicating azimuth:

```
gdaldem aspect <input_dem> <output_aspect_map>
            [-trigonometric] [-zero_for_flat]
            [-alg ZevenbergenThorne]
            [-compute_edges] [-b <band>] [-of format] [-co <NAME>=<VALUE>]... [-q]
```

Generate a color relief map:

```
gdaldem color-relief <input_dem> <color_text_file> <output_color_relief_map>
             [-alpha] [-exact_color_entry | -nearest_color_entry]
             [-b <band>] [-of format] [-co <NAME>=<VALUE>]... [-q]

where color_text_file contains lines of the format "elevation_value red green blue␣
↪[alpha]". If alpha column is present it can be enabled for use with '-alpha'.
```

Generate a Terrain Ruggedness Index (TRI) map:

```
gdaldem TRI input_dem output_TRI_map
            [-alg Wilson|Riley]
            [-compute_edges] [-b Band (default=1)] [-of format] [-q]
```

Generate a Topographic Position Index (TPI) map:

```
gdaldem TPI <input_dem> <output_TPI_map>
            [-compute_edges] [-b <band>] [-of <format>] [-co <NAME>=<VALUE>]... [-q]
```

Generate a roughness map:

```
gdaldem roughness <input_dem> <output_roughness_map>
            [-compute_edges] [-b <band>] [-of <format>] [-co <NAME>=<VALUE>]... [-q]
```

Description

The **gdaldem** generates output rasters using different algorithms for terrain analysis.

In general, it assumes that x, y and z units are identical. However, for hillshade and slope computation, starting with GDAL 3.11, if none of *-scale*, *-xscale* and *-yscale* are specified, and the CRS is a geographic or projected CRS, **gdaldem** will automatically determine the appropriate ratio from the units of the CRS, as well as the potential value of the units of the raster band (as returned by *GDALRasterBand::GetUnitType()*, if it is metre, foot international or US survey foot). Note that for geographic CRS, the result for source datasets at high latitudes may be incorrect, and prior reprojection to a polar projection might be needed.

For hillshade and slope computation, if x (east-west) and y (north-south) units are identical, but z (elevation) units are different, the *-scale* option (or *-xscale* and *-yscale*) can be used to set the ratio of vertical units to horizontal. For LatLong projections near the equator, where units of latitude and units of longitude are similar, elevation (z) units can be converted to be compatible by using scale=370400 (if elevation is in feet) or scale=111120 (if elevation is in meters). For locations not near the equator, it would be best to reproject your grid using gdalwarp before using gdaldem.

<mode>

Where <mode> is one of the seven available modes:

- `hillshade`

  Generate a shaded relief map from any GDAL-supported elevation raster.

- `slope`

  Generate a slope map from any GDAL-supported elevation raster.

- `aspect`

  Generate an aspect map from any GDAL-supported elevation raster.

- `color-relief`

  Generate a color relief map from any GDAL-supported elevation raster.

- `TRI`

  Generate a map of Terrain Ruggedness Index from any GDAL-supported elevation raster.

- `TPI`

  Generate a map of Topographic Position Index from any GDAL-supported elevation raster.

- `roughness`

  Generate a map of roughness from any GDAL-supported elevation raster.

Tip

Equivalent in new "gdal" command line interface:

- *gdal raster aspect*
- *gdal raster color-map*
- *gdal raster hillshade*
- *gdal raster slope*
- *gdal raster tpi*
- *gdal raster tri*

The following general options are available:

--help
Show this help message and exit

--help-general
Gives a brief usage message for the generic GDAL commandline options and exit.

<input_dem>
The input DEM raster to be processed

<output_xxx_map>
The output raster produced

-of <format>
Select the output format.

If not specified, the format is guessed from the extension (previously was *GTiff -- GeoTIFF File Format*). Use the short format name.

-compute_edges
Do the computation at raster edges and near nodata values

-b <band>
Select an input band to be processed. Bands are numbered from 1.

-co <NAME>=<VALUE>
Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--format* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

-q, --quiet
Suppress progress messages and other non-error output.

For all algorithms, except color-relief, a nodata value in the target dataset will be emitted if at least one pixel set to the nodata value is found in the 3x3 window centered around each source pixel. The consequence is that there will be a 1-pixel border around each image set with nodata value.

If `-compute_edges` is specified, gdaldem will compute values at image edges or if a nodata value is found in the 3x3 window, by interpolating missing values.

Modes

hillshade

This command outputs an 8-bit raster with a nice shaded relief effect. It's very useful for visualizing the terrain. You can optionally specify the azimuth and altitude of the light source, a vertical exaggeration factor and a scaling factor to account for differences between vertical and horizontal units.

The value 0 is used as the output nodata value.

The following specific options are available :

-alg `Horn|ZevenbergenThorne`

The literature suggests Zevenbergen & Thorne to be more suited to smooth landscapes, whereas Horn's formula to perform better on rougher terrain.

-z `<factor>`

Vertical exaggeration used to pre-multiply the elevations

-s `<scale>`

Ratio of vertical units to horizontal units. If the horizontal unit of the source DEM is degrees (e.g Lat/Long WGS84 projection), you can use scale=111120 if the vertical units are meters (or scale=370400 if they are in feet).

Starting with GDAL 3.11, if none of *-scale*, *-xscale* and *-yscale* are specified, and the CRS is a geographic or projected CRS, **gdaldem** will automatically determine the appropriate ratio from the units of the CRS, as well as the potential value of the units of the raster band (as returned by *GDALRasterBand::GetUnitType()*, if it is metre, foot international or US survey foot). Note that for geographic CRS, the result for source datasets at high latitudes may be incorrect, and prior reprojection to a polar projection might be needed.

The effect of this option is the same as specifying *-xscale* and *-yscale* with the same value as *-scale*. *-scale* is mutually exclusive with *-xscale* and *-yscale*

-xscale `<scale>`

Added in version 3.11.

Ratio of vertical units to horizontal X axis units. If the horizontal unit of the source DEM is degrees (e.g Lat/Long WGS84 projection), you can use scale=111120 if the vertical units are meters (or scale=370400 if they are in feet).

Starting with GDAL 3.11, if none of *-scale*, *-xscale* and *-yscale* are specified, and the CRS is a geographic or projected CRS, **gdaldem** will automatically determine the appropriate ratio from the units of the CRS, as well as the potential value of the units of the raster band (as returned by *GDALRasterBand::GetUnitType()*, if it is metre, foot international or US survey foot). Note that for geographic CRS, the result for source datasets at high latitudes may be incorrect, and prior reprojection to a polar projection might be needed.

If *-xscale* is specified, *-yscale* must also be specified.

-yscale `<scale>`

Added in version 3.11.

Ratio of vertical units to horizontal Y axis units. If the horizontal unit of the source DEM is degrees (e.g Lat/Long WGS84 projection), you can use scale=111120 if the vertical units are meters (or scale=370400 if they are in feet)

Starting with GDAL 3.11, if none of *-scale*, *-xscale* and *-yscale* are specified, and the CRS is a geographic or projected CRS, **gdaldem** will automatically determine the appropriate ratio from the units of the CRS, as well as the potential value of the units of the raster band (as returned by *GDALRasterBand::GetUnitType()*, if it is metre, foot international or US survey foot). Note that for geographic CRS, the result for source datasets at high latitudes may be incorrect, and prior reprojection to a polar projection might be needed.

If *-yscale* is specified, *-xscale* must also be specified.

-az <azimuth>

Azimuth of the light, in degrees. 0 if it comes from the top of the raster, 90 from the east, ... The default value, 315, should rarely be changed as it is the value generally used to generate shaded maps.

-alt <altitude>

Altitude of the light, in degrees. 90 if the light comes from above the DEM, 0 if it is raking light.

-combined

combined shading, a combination of slope and oblique shading.

-multidirectional

multidirectional shading, a combination of hillshading illuminated from 225 deg, 270 deg, 315 deg, and 360 deg azimuth.

-igor

shading which tries to minimize effects on other map features beneath. Can't be used with -alt option.

Added in version 3.0.

Multidirectional hillshading applies the formula of http://pubs.usgs.gov/of/1992/of92-422/of92-422.pdf.

Igor's hillshading uses formula from Maperitive http://maperitive.net/docs/Commands/GenerateReliefImageIgor.html.

slope

This command will take a DEM raster and output a 32-bit float raster with slope values. You have the option of specifying the type of slope value you want: degrees or percent slope. In cases where the horizontal units differ from the vertical units, you can also supply a scaling factor.

The value *-9999* is used as the output nodata value.

The following specific options are available :

-alg Horn|ZevenbergenThorne

The literature suggests Zevenbergen & Thorne to be more suited to smooth landscapes, whereas Horn's formula to perform better on rougher terrain.

-p

If specified, the slope will be expressed as percent slope. Otherwise, it is expressed as degrees

-s <scale>

Ratio of vertical units to horizontal units. If the horizontal unit of the source DEM is degrees (e.g Lat/Long WGS84 projection), you can use scale=111120 if the vertical units are meters (or scale=370400 if they are in feet).

Starting with GDAL 3.11, if none of *-scale*, *-xscale* and *-yscale* are specified, and the CRS is a geographic or projected CRS, **gdaldem** will automatically determine the appropriate ratio from the units of the CRS, as well as the potential value of the units of the raster band (as returned by *GDALRasterBand::GetUnitType()*, if it is metre, foot international or US survey foot). Note that for geographic CRS, the result for source datasets at high latitudes may be incorrect, and prior reprojection to a polar projection might be needed.

The effect of this option is the same as specifying *-xscale* and *-yscale* with the same value as *-scale*. *-scale* is mutually exclusive with *-xscale* and *-yscale*

-xscale <scale>

Added in version 3.11.

Ratio of vertical units to horizontal X axis units. If the horizontal unit of the source DEM is degrees (e.g Lat/Long WGS84 projection), you can use scale=111120 if the vertical units are meters (or scale=370400 if they are in feet).

Starting with GDAL 3.11, if none of *-scale*, *-xscale* and *-yscale* are specified, and the CRS is a geographic or projected CRS, **gdaldem** will automatically determine the appropriate ratio from the units of the CRS, as well as the potential value of the units of the raster band (as returned by *GDALRasterBand::GetUnitType()*, if it is metre, foot international or US survey foot). Note that for geographic CRS, the result for source datasets at high latitudes may be incorrect, and prior reprojection to a polar projection might be needed.

If *-xscale* is specified, *-yscale* must also be specified.

-yscale <scale>

Added in version 3.11.

Ratio of vertical units to horizontal Y axis units. If the horizontal unit of the source DEM is degrees (e.g Lat/Long WGS84 projection), you can use scale=111120 if the vertical units are meters (or scale=370400 if they are in feet)

Starting with GDAL 3.11, if none of *-scale*, *-xscale* and *-yscale* are specified, and the CRS is a geographic or projected CRS, **gdaldem** will automatically determine the appropriate ratio from the units of the CRS, as well as the potential value of the units of the raster band (as returned by *GDALRasterBand::GetUnitType()*, if it is metre, foot international or US survey foot). Note that for geographic CRS, the result for source datasets at high latitudes may be incorrect, and prior reprojection to a polar projection might be needed.

If *-yscale* is specified, *-xscale* must also be specified.

aspect

This command outputs a 32-bit float raster with values between 0° and 360° representing the azimuth that slopes are facing. The definition of the azimuth is such that : 0° means that the slope is facing the North, 90° it's facing the East, 180° it's facing the South and 270° it's facing the West (provided that the top of your input raster is north oriented). The aspect value -9999 is used as the nodata value to indicate undefined aspect in flat areas with slope=0.

The following specifics options are available :

-alg Horn|ZevenbergenThorne

The literature suggests Zevenbergen & Thorne to be more suited to smooth landscapes, whereas Horn's formula to perform better on rougher terrain.

-trigonometric

Return trigonometric angle instead of azimuth. Thus 0° means East, 90° North, 180° West, 270° South.

-zero_for_flat

Return 0 for flat areas with slope=0, instead of -9999.

By using those 2 options, the aspect returned by gdaldem aspect should be identical to the one of GRASS r.slope.aspect. Otherwise, it's identical to the one of Matthew Perry's `aspect.cpp` utility.

color-relief

This command outputs a 3-band (RGB) or 4-band (RGBA) raster with values are computed from the elevation and a text-based color configuration file, containing the association between various elevation values and the corresponding wished color. By default, the colors between the given elevation values are blended smoothly and the result is a nice colorized DEM. The -exact_color_entry or -nearest_color_entry options can be used to avoid that linear interpolation for values that don't match an index of the color configuration file.

The following specifics options are available :

color_text_file

Text-based color configuration file

-alpha

Add an alpha channel to the output raster

-exact_color_entry

Use strict matching when searching in the color configuration file. If none matching color entry is found, the "0,0,0,0" RGBA quadruplet will be used

-nearest_color_entry

Use the RGBA quadruplet corresponding to the closest entry in the color configuration file.

The color-relief mode is the only mode that supports VRT as output format. In that case, it will translate the color configuration file into appropriate LUT elements. Note that elevations specified as percentage will be translated as absolute values, which must be taken into account when the statistics of the source raster differ from the one that was used when building the VRT.

The text-based color configuration file generally contains 4 columns per line: the elevation value and the corresponding Red, Green, Blue component (between 0 and 255). The elevation value can be any floating point value, or the nv keyword for the nodata value. The elevation can also be expressed as a percentage: 0% being the minimum value found in the raster, 100% the maximum value.

An extra column can be optionally added for the alpha component. If it is not specified, full opacity (255) is assumed.

Various field separators are accepted: comma, tabulation, spaces, ':'.

Common colors used by GRASS can also be specified by using their name, instead of the RGB triplet. The supported list is: white, black, red, green, blue, yellow, magenta, cyan, aqua, grey/gray, orange, brown, purple/violet and indigo.

GMT `.cpt` palette files are also supported (COLOR_MODEL = RGB only).

Note: the syntax of the color configuration file is derived from the one supported by GRASS r.colors utility. ESRI HDR color table files (.clr) also match that syntax. The alpha component and the support of tab and comma as separators are GDAL specific extensions.

For example:

```
3500   white
2500   235:220:175
50%   190 185 135
700    240 250 150
0      50  180  50
nv     0   0   0   0
```

To implement a "round to the floor value" mode, the elevation value can be duplicate with a new value being slightly above the threshold. For example to have red in [0,10], green in]10,20] and blue in]20,30]:

```
0       red
10      red
10.001  green
20      green
20.001  blue
30      blue
```

TRI

This command outputs a single-band raster with values computed from the elevation. *TRI* stands for Terrain Ruggedness Index, which measures the difference between a central pixel and its surrounding cells.

The value -9999 is used as the output nodata value.

The following option is available:

-alg `Wilson|Riley`

Starting with GDAL 3.3, the Riley algorithm (see Riley, S.J., De Gloria, S.D., Elliot, R. (1999): A Terrain Ruggedness that Quantifies Topographic Heterogeneity. Intermountain Journal of Science, Vol.5, No.1-4, pp.23-27) is available and the new default value. This algorithm uses the square root of the sum of the square of the difference between a central pixel and its surrounding cells. This is recommended for terrestrial use cases.

The Wilson (see Wilson et al 2007, Marine Geodesy 30:3-35) algorithm uses the mean difference between a central pixel and its surrounding cells. This is recommended for bathymetric use cases.

TPI

This command outputs a single-band raster with values computed from the elevation. *TPI* stands for Topographic Position Index, which is defined as the difference between a central pixel and the mean of its surrounding cells (see Wilson et al 2007, Marine Geodesy 30:3-35).

The value -9999 is used as the output nodata value.

There are no specific options.

roughness

This command outputs a single-band raster with values computed from the elevation. Roughness is the largest inter-cell difference of a central pixel and its surrounding cell, as defined in Wilson et al (2007, Marine Geodesy 30:3-35).

The value -9999 is used as the output nodata value.

There are no specific options.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

C API

This utility is also callable from C with *`GDALDEMProcessing()`*.

Authors

Matthew Perry perrygeo@gmail.com, Even Rouault even.rouault@spatialys.com, Howard Butler hobu.inc@gmail.com, Chris Yesson chris.yesson@ioz.ac.uk

Derived from code by Michael Shapiro, Olga Waupotitsch, Marjorie Larson, Jim Westervelt: U.S. Army CERL, 1993. GRASS 4.1 Reference Manual. U.S. Army Corps of Engineers, Construction Engineering Research Laboratories, Champaign, Illinois, 1-425.

See also

Documentation of related GRASS utilities:

https://grass.osgeo.org/grass84/manuals/r.slope.aspect.html

https://grass.osgeo.org/grass84/manuals/r.relief.html

https://grass.osgeo.org/grass84/manuals/r.colors.html

5.2.2.26 gdalenhance

Synopsis

```
gdalenhance [--help-general]
            [-of format] [-co "NAME=VALUE"]*
            [-ot {Byte/Int16/UInt16/UInt32/Int32/Float32/Float64/
                    CInt16/CInt32/CFloat32/CFloat64}]
            [-equalize]
            [-config filename]
            <src_raster> [<dst_raster>]
```

Description

The **gdalenhance** utility enhances the contrast of raster images using histogram equalization. It can either generate a LUT (look-up-table) that can later be applied to enhance contrast, or it can write the enhanced image directly.

Fig. 17: Original image.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-of format

Select the output format. The default is GeoTIFF (GTiff).

VRT output is not supported.

-co <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file

Fig. 18: Image after histogram equalization using **gdalenhance**.

created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--format* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

-ot <type>

Force the output image bands to have a specific data type supported by the driver, which may be one of the following: `Byte`, `Int8`, `UInt16`, `Int16`, `UInt32`, `Int32`, `UInt64`, `Int64`, `Float32`, `Float64`, `CInt16`, `CInt32`, `CFloat32` or `CFloat64`.

-equalize

Get source image histogram and compute equalization LUTs from it.

-config <filename>

Apply a specified LUT to enhance the contrast of the image. The number of lines in the LUT file should be equal to the number of bands of the images. Example of LUT file:

```
1:Band -0.5:ScaleMin 255.5:ScaleMax 0 0 8 16 16 17 17 17 17 18 18 18 18 18 18 19 19␣
↪19 19 20 20 20 21 21 21 22 22 22 23 23 24 24 25 25 26 27 27 28 29 30 30 31 32 33␣
↪34 35 36 37 38 39 40 42 43 44 46 47 48 50 51 53 55 56 58 60 62 63 65 67 69 71 73␣
↪75 77 79 81 83 85 87 89 91 93 95 97 99 101 103 105 107 109 111 113 115 117 119␣
↪121 123 125 127 129 131 133 135 137 138 140 142 144 146 148 150 152 153 155 157␣
↪159 161 162 164 166 168 169 171 173 174 176 178 179 181 182 184 186 187 189 190␣
↪192 193 195 196 198 199 200 202 203 205 206 207 208 210 211 212 213 214 216 217␣
↪218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 234 235 236␣
↪237 238 239 239 240 241 241 242 243 244 244 245 245 246 246 247 247 248 248 249␣
↪249 250 250 250 251 251 251 251 251 252 252 252 252 252 252 252 252 252 252 253␣
↪253 253 253 253 253 253 253 253 253 253 253 253 253 253 254 254 254 254 254 254␣
↪255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255␣
↪255 255
2:Band -0.5:ScaleMin 255.5:ScaleMax 0 0 8 16 16 16 17 17 17 17 17 17 17 17 17 18 18␣
↪18 18 18 18 19 19 19 19 19 20 20 20 20 21 21 22 22 23 23 24 24 25 26 27 27 28 29␣
↪30 31 32 33 34 35 36 37 38 39 41 42 43 45 46 47 49 51 52 54 55 57 59 60 62 64 66␣
↪67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99 101 103 105 108 110 112 114␣
↪116 118 120 122 124 126 128 130 132 134 136 138 140 142 144 146 148 150 152 154␣
↪156 157 159 161 163 165 166 168 170 172 173 175 177 179 180 182 183 185 187 188␣
```

(continues on next page)

(continued from previous page)

```
↪190 191 193 194 196 197 199 200 202 203 204 206 207 208 209 211 212 213 214 215␣
↪216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 230 231 232 233 234␣
↪235 235 236 237 238 239 239 240 241 241 242 243 243 244 245 245 246 246 247 247␣
↪248 248 248 249 249 250 250 250 250 251 251 251 251 252 252 252 252 252 252 252␣
↪252 252 252 253 253 253 253 253 253 253 253 253 253 253 253 253 253 253 253 253␣
↪254 254 254 254 254 254 255 255 255 255 255 255 255 255 255 255 255 255 255 255␣
↪255
3:Band -0.5:ScaleMin 255.5:ScaleMax 0 0 9 17 17 18 18 18 18 18 19 19 19 19 19 19 20␣
↪20 20 20 20 20 21 21 21 21 21 21 22 22 22 22 22 23 23 23 24 24 24 25 25 26 26 27␣
↪28 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 44 45 46 47 48 50 51 53 54 56 57␣
↪59 60 62 63 65 67 69 70 72 74 76 78 80 82 84 86 88 90 92 94 97 99 101 103 105 107␣
↪109 111 113 115 116 118 120 122 124 126 127 129 131 133 134 136 138 139 141 143␣
↪144 146 147 149 151 152 154 155 157 159 160 162 163 165 167 168 170 171 173 175␣
↪176 178 179 181 182 184 186 187 189 190 192 193 195 197 198 200 201 203 204 206␣
↪207 209 210 212 213 214 216 217 218 219 220 222 223 224 225 226 227 228 229 230␣
↪231 232 233 233 234 235 236 237 238 239 239 240 241 242 243 243 244 245 246 246␣
↪247 247 248 248 249 249 250 250 251 251 251 251 252 252 252 252 252 252 252 252␣
↪253 253 253 253 253 253 253 253 253 253 253 253 253 253 253 253 253 253 253 253␣
↪253 253 253 253 253 253 253 253 253 253 254 254 255 255 255 255 255 255 2550
```

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Apply equalization histogram to enhance the contrast of the image

```
gdalenhance -equalize rgb.tif rgb_equalize_enhance.tif
```

Example 2: Write an equalization LUT to the console

```
gdalenhance -equalize rgb.tif
```

Example 3: Apply a custom LUT (look up-table) to enhance the contrast of the image

```
gdalenhance -config enhance_config rgb.tif rgb_custom_enhance.tif
```

5.2.2.27 gdalinfo

Synopsis

```
Usage: gdalinfo [--help] [--long-usage] [--help-general]
                [-json] [-mm]
                [[-stats]|[-approx_stats]]
                [-hist]
                [-nogcp] [-nomd] [-norat] [-noct] [-nofl] [-nonodata] [-nomask]
                [-checksum] [-listmdd] [-proj4]
                [-wkt_format <WKT1|WKT1_ESRI|WKT2|WKT2_2015|WKT2_2018|WKT2_2019>]
```

(continues on next page)

(continued from previous page)

```
                [-sd <n>] [-oo <NAME>=<VALUE>]... [-if <format>]...
                [-mdd <domain>|all]
                <dataset_name>
```

Description

gdalinfo program lists various information about a GDAL supported raster dataset.

The following command line parameters can appear in any order

> **Tip**
>
> Equivalent in new "gdal" command line interface:
>
> See *gdal raster info*.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-json

Display the output in json format. Since GDAL 3.6, this includes key-value pairs useful for building a STAC item , including statistics and histograms if `-stats` or `-hist` flags are passed, respectively.

-mm

Force computation of the actual min/max values for each band in the dataset.

-stats

Read and display image statistics. Force computation if no statistics are stored in an image.

-approx_stats

Read and display image statistics. Force computation if no statistics are stored in an image. However, they may be computed based on overviews or a subset of all tiles. Useful if you are in a hurry and don't need precise stats.

-hist

Report histogram information for all bands.

-nogcp

Suppress ground control points list printing. It may be useful for datasets with huge amount of GCPs, such as L1B AVHRR or HDF4 MODIS which contain thousands of them.

-nomd

Suppress metadata printing. Some datasets may contain a lot of metadata strings.

-norat

Suppress printing of raster attribute table.

-noct

Suppress printing of color table.

-nonodata

Added in version 3.10.

Suppress nodata printing. Implies *-nomask*.

Can be useful for example when querying a remove GRIB2 dataset that has an index .idx side-car file, together with *-nomd*

-nomask

Added in version 3.10.

Suppress band mask printing. Is implied if *-nonodata* is specified.

-checksum

Force computation of the checksum for each band in the dataset.

-listmdd

List all metadata domains available for the dataset.

-mdd `<domain>|all`

adds metadata using:

`domain` Report metadata for the specified domain.

`all` Report metadata for all domains.

-nofl

Only display the first file of the file list.

-wkt_format `WKT1|WKT1_ESRI|WKT2|WKT2_2015|WKT2_2018|WKT2_2019`

WKT format used to display the SRS. Currently the supported values are:

`WKT1`

`WKT1_ESRI`

`WKT2` (latest WKT version, currently *WKT2_2019*)

`WKT2_2015`

`WKT2_2018` (deprecated)

`WKT2_2019`

Added in version 3.0.0.

-sd `<n>`

If the input dataset contains several subdatasets read and display a subdataset with specified `n` number (starting from 1). This is an alternative of giving the full subdataset name.

-proj4

Report a PROJ.4 string corresponding to the file's coordinate system.

-oo `<NAME>=<VALUE>`

Dataset open option (format specific).

-if `<format>`

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

Added in version 3.2.

The gdalinfo will report all of the following (if known):

- The format driver used to access the file.
- Raster size (in pixels and lines).
- The coordinate system for the file (in OGC WKT).
- The geotransform associated with the file (rotational coefficients are currently not reported).
- Corner coordinates in georeferenced, and if possible lat/long based on the full geotransform (but not GCPs).
- Ground control points.
- File wide (including subdatasets) metadata.
- Band data types.
- Band color interpretations.
- Band block size.
- Band descriptions.
- Band min/max values (internally known and possibly computed).
- Band checksum (if computation asked).
- Band NODATA value.
- Band overview resolutions available.
- Band unit type (i.e.. "meters" or "feet" for elevation bands).
- Band pseudo-color tables.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

C API

This utility is also callable from C with *GDALInfo()*.

Examples

Example 1: Default output format

```
$ gdalinfo utmsmall.tif
Driver: GTiff/GeoTIFF
Files: utmsmall.tif
Size is 100, 100
Coordinate System is:
PROJCRS["NAD27 / UTM zone 11N",
    BASEGEOGCRS["NAD27",
        DATUM["North American Datum 1927",
            ELLIPSOID["Clarke 1866",6378206.4,294.978698213898,
                LENGTHUNIT["metre",1]]],
        PRIMEM["Greenwich",0,
            ANGLEUNIT["degree",0.0174532925199433]],
        ID["EPSG",4267]],
```

(continues on next page)

(continued from previous page)

```
    CONVERSION["Transverse Mercator",
        METHOD["Transverse Mercator",
            ID["EPSG",9807]],
        PARAMETER["Latitude of natural origin",0,
            ANGLEUNIT["degree",0.0174532925199433],
            ID["EPSG",8801]],
        PARAMETER["Longitude of natural origin",-117,
            ANGLEUNIT["degree",0.0174532925199433],
            ID["EPSG",8802]],
        PARAMETER["Scale factor at natural origin",0.9996,
            SCALEUNIT["unity",1],
            ID["EPSG",8805]],
        PARAMETER["False easting",500000,
            LENGTHUNIT["metre",1],
            ID["EPSG",8806]],
        PARAMETER["False northing",0,
            LENGTHUNIT["metre",1],
            ID["EPSG",8807]]],
    CS[Cartesian,2],
        AXIS["easting",east,
            ORDER[1],
            LENGTHUNIT["metre",1]],
        AXIS["northing",north,
            ORDER[2],
            LENGTHUNIT["metre",1]],
    ID["EPSG",26711]]
Data axis to CRS axis mapping: 1,2
Origin = (440720.000000000000000,3751320.000000000000000)
Pixel Size = (60.000000000000000,-60.000000000000000)
Metadata:
  AREA_OR_POINT=Area
Image Structure Metadata:
  INTERLEAVE=BAND
Corner Coordinates:
Upper Left  (  440720.000, 3751320.000) (117d38'28.21"W, 33d54' 8.47"N)
Lower Left  (  440720.000, 3745320.000) (117d38'26.75"W, 33d50'53.66"N)
Upper Right (  446720.000, 3751320.000) (117d34'34.59"W, 33d54' 9.62"N)
Lower Right (  446720.000, 3745320.000) (117d34'33.28"W, 33d50'54.82"N)
Center      (  443720.000, 3748320.000) (117d36'30.71"W, 33d52'31.66"N)
Band 1 Block=100x81 Type=Byte, ColorInterp=Gray
```

Example 2: JSON output

For corner coordinates formatted as decimal degree instead of the above degree, minute, second, inspect the `wgs84Extent` member of `gdalinfo -json`:

Example of JSON output with `gdalinfo -json utmsmall.tif`:

```
{
  "description":"utmsmall.tif",
  "driverShortName":"GTiff",
  "driverLongName":"GeoTIFF",
```

(continues on next page)

(continued from previous page)


```
  "files":[
    "utmsmall.tif"
  ],
  "size":[
    100,
    100
  ],
  "coordinateSystem":{
    "wkt":"PROJCRS[\"NAD27 / UTM zone 11N\",\n    BASEGEOGCRS[\"NAD27\",\n        DATUM[\
↪"North American Datum 1927\",\n            ELLIPSOID[\"Clarke 1866\",6378206.4,294.
↪978698213898,\n                LENGTHUNIT[\"metre\",1]]],\n        PRIMEM[\"Greenwich\
↪",0,\n            ANGLEUNIT[\"degree\",0.0174532925199433]],\n        ID[\"EPSG\",
↪4267]],\n    CONVERSION[\"Transverse Mercator\",\n        METHOD[\"Transverse Mercator\
↪",\n            ID[\"EPSG\",9807]],\n        PARAMETER[\"Latitude of natural origin\",
↪0,\n            ANGLEUNIT[\"degree\",0.0174532925199433],\n            ID[\"EPSG\",
↪8801]],\n        PARAMETER[\"Longitude of natural origin\",-117,\n            ␣
↪ANGLEUNIT[\"degree\",0.0174532925199433],\n            ID[\"EPSG\",8802]],\n        ␣
↪PARAMETER[\"Scale factor at natural origin\",0.9996,\n            SCALEUNIT[\"unity\",
↪1],\n            ID[\"EPSG\",8805]],\n        PARAMETER[\"False easting\",500000,\n    ␣
↪        LENGTHUNIT[\"metre\",1],\n            ID[\"EPSG\",8806]],\n        PARAMETER[\
↪"False northing\",0,\n            LENGTHUNIT[\"metre\",1],\n            ID[\"EPSG\",
↪8807]]],\n    CS[Cartesian,2],\n        AXIS[\"easting\",east,\n            ORDER[1],\
↪n            LENGTHUNIT[\"metre\",1]],\n        AXIS[\"northing\",north,\n            ␣
↪ORDER[2],\n            LENGTHUNIT[\"metre\",1]],\n    ID[\"EPSG\",26711]]",
    "dataAxisToSRSAxisMapping":[
      1,
      2
    ]
  },
  "geoTransform":[
    440720.0,
    60.0,
    0.0,
    3751320.0,
    0.0,
    -60.0
  ],
  "metadata":{
    "":{
      "AREA_OR_POINT":"Area"
    },
    "IMAGE_STRUCTURE":{
      "INTERLEAVE":"BAND"
    }
  },
  "cornerCoordinates":{
    "upperLeft":[
      440720.0,
      3751320.0
    ],
    "lowerLeft":[
      440720.0,
```


(continues on next page)

(continued from previous page)


```
      3745320.0
    ],
    "lowerRight":[
      446720.0,
      3745320.0
    ],
    "upperRight":[
      446720.0,
      3751320.0
    ],
    "center":[
      443720.0,
      3748320.0
    ]
  },
  "wgs84Extent":{
    "type":"Polygon",
    "coordinates":[
      [
        [
          -117.6420428,
          33.9023684
        ],
        [
          -117.6416377,
          33.848257
        ],
        [
          -117.5767844,
          33.8485775
        ],
        [
          -117.5771485,
          33.9026895
        ],
        [
          -117.6420428,
          33.9023684
        ]
      ]
    ]
  },
  "bands":[
    {
      "band":1,
      "block":[
        100,
        81
      ],
      "type":"Byte",
      "colorInterpretation":"Gray",
      "metadata":{
```


(continues on next page)

(continued from previous page)


```
      }
    }
  ],
  "stac":{
    "proj:shape":[
      100,
      100
    ],
    "proj:wkt2":"PROJCRS[\"NAD27 / UTM zone 11N\",\n    BASEGEOGCRS[\"NAD27\",\n
↪DATUM[\"North American Datum 1927\",\n            ELLIPSOID[\"Clarke 1866\",6378206.4,
↪294.978698213898,\n                LENGTHUNIT[\"metre\",1]]],\n        PRIMEM[\
↪"Greenwich\",0,\n            ANGLEUNIT[\"degree\",0.0174532925199433]],\n        ID[\
↪"EPSG\",4267]],\n    CONVERSION[\"Transverse Mercator\",\n        METHOD[\"Transverse
↪Mercator\",\n            ID[\"EPSG\",9807]],\n        PARAMETER[\"Latitude of natural
↪origin\",0,\n            ANGLEUNIT[\"degree\",0.0174532925199433],\n            ID[\
↪"EPSG\",8801]],\n        PARAMETER[\"Longitude of natural origin\",-117,\n
↪ANGLEUNIT[\"degree\",0.0174532925199433],\n            ID[\"EPSG\",8802]],\n
↪PARAMETER[\"Scale factor at natural origin\",0.9996,\n            SCALEUNIT[\"unity\",
↪1],\n            ID[\"EPSG\",8805]],\n        PARAMETER[\"False easting\",500000,\n
↪        LENGTHUNIT[\"metre\",1],\n            ID[\"EPSG\",8806]],\n        PARAMETER[\
↪"False northing\",0,\n            LENGTHUNIT[\"metre\",1],\n            ID[\"EPSG\",
↪8807]]],\n    CS[Cartesian,2],\n        AXIS[\"easting\",east,\n            ORDER[1],\
↪n            LENGTHUNIT[\"metre\",1]],\n        AXIS[\"northing\",north,\n
↪ORDER[2],\n            LENGTHUNIT[\"metre\",1]],\n    ID[\"EPSG\",26711]]",
    "proj:epsg":26711,
    "proj:projjson":{
      "$schema":"https://proj.org/schemas/v0.7/projjson.schema.json",
      "type":"ProjectedCRS",
      "name":"NAD27 / UTM zone 11N",
      "base_crs":{
        "name":"NAD27",
        "datum":{
          "type":"GeodeticReferenceFrame",
          "name":"North American Datum 1927",
          "ellipsoid":{
            "name":"Clarke 1866",
            "semi_major_axis":6378206.4,
            "inverse_flattening":294.978698213898
          }
        },
        "coordinate_system":{
          "subtype":"ellipsoidal",
          "axis":[
            {
              "name":"Geodetic latitude",
              "abbreviation":"Lat",
              "direction":"north",
              "unit":"degree"
            },
            {
              "name":"Geodetic longitude",
              "abbreviation":"Lon",
```


(continues on next page)

(continued from previous page)

```
            "direction":"east",
            "unit":"degree"
          }
        ]
      },
      "id":{
        "authority":"EPSG",
        "code":4267
      }
    },
    "conversion":{
      "name":"Transverse Mercator",
      "method":{
        "name":"Transverse Mercator",
        "id":{
          "authority":"EPSG",
          "code":9807
        }
      },
      "parameters":[
        {
          "name":"Latitude of natural origin",
          "value":0,
          "unit":"degree",
          "id":{
            "authority":"EPSG",
            "code":8801
          }
        },
        {
          "name":"Longitude of natural origin",
          "value":-117,
          "unit":"degree",
          "id":{
            "authority":"EPSG",
            "code":8802
          }
        },
        {
          "name":"Scale factor at natural origin",
          "value":0.9996,
          "unit":"unity",
          "id":{
            "authority":"EPSG",
            "code":8805
          }
        },
        {
          "name":"False easting",
          "value":500000,
          "unit":"metre",
          "id":{
```

(continues on next page)

(continued from previous page)


```
            "authority":"EPSG",
            "code":8806
          }
        },
        {
          "name":"False northing",
          "value":0,
          "unit":"metre",
          "id":{
            "authority":"EPSG",
            "code":8807
          }
        }
      ]
    },
    "coordinate_system":{
      "subtype":"Cartesian",
      "axis":[
        {
          "name":"Easting",
          "abbreviation":"",
          "direction":"east",
          "unit":"metre"
        },
        {
          "name":"Northing",
          "abbreviation":"",
          "direction":"north",
          "unit":"metre"
        }
      ]
    },
    "id":{
      "authority":"EPSG",
      "code":26711
    }
  },
  "proj:transform":[
    60.0,
    0.0,
    440720.0,
    0.0,
    -60.0,
    3751320.0
  ],
  "raster:bands":[
    {
      "data_type":"uint8"
    }
  ],
  "eo:bands":[
    {
```


(continues on next page)

(continued from previous page)

```
        "name":"b1",
        "description":"Gray"
      }
    ]
  }
}
```

5.2.2.28 gdallocationinfo

Synopsis

```
Usage: gdallocationinfo [--help] [--long-usage] [--help-general]
                        [-xml] [-lifonly] [-valonly] [-E] [-field_sep <sep>]
                        [-ignore_extra_input] [-b <band>]...
                        [-overview <overview_level>]
                        [-r nearest|bilinear|cubic|cubicspline]
                        [[-l_srs <srs_def>]|[-geoloc]|[-wgs84]]
                        [-oo <NAME>=<VALUE>]...
                        <srcfile> <x> <y>
```

Description

The **gdallocationinfo** utility provide a mechanism to query information about a pixel given its location in one of a variety of coordinate systems. Several reporting options are provided.

> **Tip**
>
> Equivalent in new "gdal" command line interface:
>
> See *gdal raster pixel-info*.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-xml

The output report will be XML formatted for convenient post processing. (*Example 2*).

-lifonly

The only output is filenames production from the LocationInfo request against the database (i.e. for identifying impacted file from VRT).

-valonly

The only output is the pixel values of the selected pixel on each of the selected bands. By default, the value of each band is output on a separate line, unless *-field_sep* is specified.

-b <band>

Selects a band to query. Multiple bands can be listed. By default all bands are queried.

-r {nearest|bilinear|cubic|cubicspline}

Added in version 3.10.

Select a sampling algorithm. The default is `nearest`.

The available methods are:

`nearest` applies a nearest neighbour.

`bilinear` applies a bilinear convolution kernel.

`cubic` applies a cubic convolution kernel.

`cubicspline` applies a B-Spline convolution kernel.

-overview <overview_level>

Query the (overview_level)th overview (overview_level=1 is the 1st overview), instead of the base band. Note that the x,y location (if the coordinate system is pixel/line) must still be given with respect to the base band.

-l_srs <srs_def>

The coordinate system of the input x, y location.

-geoloc

Indicates input x,y points are in the georeferencing system of the image.

-wgs84

Indicates input x,y points are WGS84 long, lat.

-oo <NAME>=<VALUE>

Dataset open option (format specific)

-ignore_extra_input

Added in version 3.9.

Set this flag to avoid extra non-numeric content at end of input lines to be appended to the output lines in -valonly mode (requires *-field_sep* to be also defined), or as a dedicated field in default or *-xml* modes.

-E

Added in version 3.9.

Enable Echo mode, where input coordinates are prepended to the output lines in *-valonly* mode.

-field_sep <sep>

Added in version 3.9.

Defines the field separator, used in *-valonly* mode, to separate different values. It defaults to the new-line character, which means that when querying a raster with several bands, the output will contain one value per line, which may make it hard to recognize which value belongs to which set of input x,y points when several ones are provided. Defining the field separator is also needed

<srcfile>

The source GDAL raster datasource name.

<x>

X location of target pixel. By default the coordinate system is pixel/line unless -l_srs, -wgs84 or -geoloc supplied.

<y>

Y location of target pixel. By default the coordinate system is pixel/line unless -l_srs, -wgs84 or -geoloc supplied.

This utility is intended to provide a variety of information about a pixel. Currently it reports:

- The location of the pixel in pixel/line space.

- The result of a LocationInfo metadata query against the datasource. This is implemented for VRT files which will report the file(s) used to satisfy requests for that pixel, and by the *MBTiles* driver.
- The raster pixel value of that pixel for all or a subset of the bands.
- The unscaled pixel value if a Scale and/or Offset apply to the band.

The pixel selected is requested by x/y coordinate on the command line, or read from stdin. More than one coordinate pair can be supplied when reading coordinates from stdin. By default integer pixel/line coordinates are expected. However with use of the `-geoloc`, `-wgs84`, or `-l_srs` switches it is possible to specify the location in other coordinate systems.

The default report is in a human readable text format. It is possible to instead request xml output with the -xml switch.

For scripting purposes, the -valonly and -lifonly switches are provided to restrict output to the actual pixel values, or the LocationInfo files identified for the pixel.

It is anticipated that additional reporting capabilities will be added to gdallocationinfo in the future.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Reporting on pixel (256,256) on the file utm.tif

```
$ gdallocationinfo utm.tif 256 256
Report:
Location: (256P,256L)
Band 1:
    Value: 115
```

Example 2: Querying a VRT file providing the location in WGS84, and getting the result in XML

```
$ gdallocationinfo -xml -wgs84 utm.vrt -117.5 33.75
<Report pixel="217" line="282">
    <BandReport band="1">
        <LocationInfo>
        <File>utm.tif</File>
        </LocationInfo>
        <Value>16</Value>
    </BandReport>
</Report>
```

Example 3: Reading locations from stdin

```
$ cat coordinates.txt
443020 3748359
441197 3749005
443852 3747743

$ cat coordinates.txt | gdallocationinfo -geoloc utmsmall.tif
Report:
```

(continues on next page)

(continued from previous page)

```
  Location: (38P,49L)
  Band 1:
    Value: 214
Report:
  Location: (7P,38L)
  Band 1:
    Value: 107
Report:
  Location: (52P,59L)
  Band 1:
    Value: 148

$ cat coordinates.txt | gdallocationinfo -geoloc -valonly -E -field_sep , utmsmall.tif
443020,3748359,214
441197,3749005,107
443852,3747743,148
```

5.2.2.29 gdalmanage

Synopsis

```
Usage: gdalmanage [--help] [--help-general]
                  <mode> [-r] [-fr] [-u] [-f <format>]
                  <datasetname> [<newdatasetname>]
```

Description

The **gdalmanage** program can perform various operations on dataset files, depending on the chosen *mode*. This includes identifying dataset types and deleting, renaming or copying the files.

<mode>

Mode of operation

identify <*datasetname*>:
List data format of file(s).

copy <*datasetname*> <*newdatasetname*>:
Create a copy of the dataset file with a new name.

rename <*datasetname*> <*newdatasetname*>:
Change the name of the dataset file.

delete <*datasetname*>:
Delete dataset file(s).

-r

Recursively scan files/folders for dataset files.

-fr

Recursively scan folders for dataset files, forcing recursion in folders recognized as valid formats.

-u

Report failures if file type is unidentified.

-f <format>

Specify format of dataset file if unknown by the application. Uses short data format name (e.g. *GTiff*).

<datasetname>

Raster file to operate on. With **identify** may be repeated for multiple files.

<newdatasetname>

For copy and rename modes, you provide a *source* filename and a *target* filename, just like copy and move commands in an operating system.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Using identify mode

Report the data format of the dataset file by using the *identify* mode and specifying a data file name:

```
$ gdalmanage identify NE1_50M_SR_W.tif

NE1_50M_SR_W.tif: GTiff
```

Recursive mode will scan subfolders and report the data format:

```
$ gdalmanage identify -r 50m_raster/

NE1_50M_SR_W/ne1_50m.jpg: JPEG
NE1_50M_SR_W/ne1_50m.png: PNG
NE1_50M_SR_W/ne1_50m_20pct.tif: GTiff
NE1_50M_SR_W/ne1_50m_band1.tif: GTiff
NE1_50M_SR_W/ne1_50m_print.png: PNG
NE1_50M_SR_W/NE1_50M_SR_W.aux: HFA
NE1_50M_SR_W/NE1_50M_SR_W.tif: GTiff
NE1_50M_SR_W/ne1_50m_sub.tif: GTiff
NE1_50M_SR_W/ne1_50m_sub2.tif: GTiff
```

Using copy mode

Copy the dataset:

```
$ gdalmanage copy NE1_50M_SR_W.tif ne1_copy.tif
```

Using rename mode

Rename dataset:

```
$ gdalmanage rename NE1_50M_SR_W.tif ne1_rename.tif
```

Using delete mode

Delete the dataset:

```
gdalmanage delete NE1_50M_SR_W.tif
```

5.2.2.30 gdalmove

Synopsis

```
gdalmove [--help] [--help-general]
            [-s_srs <srs_defn>] -t_srs <srs_defn>
            [-et <max_pixel_err>] <target_file>
```

Description

The **gdalmove** script transforms the bounds of a raster file from one coordinate system to another, and then updates the coordinate system and geotransform of the file. This is done without altering pixel values at all. It is loosely similar to using gdalwarp to transform an image but avoiding the resampling step in order to avoid image damage. It is generally only suitable for transformations that are effectively linear in the area of the file.

If no error threshold value (*-et*) is provided then the file is not actually updated, but the errors that would be incurred are reported. If *-et* is provided then the file is only modify if the apparent error being introduced is less than the indicate threshold (in pixels).

Currently the transformed geotransform is computed based on the transformation of the top left, top right, and bottom left corners. A reduced overall error could be produced using a least squares fit of at least all four corner points.

> **Note**
>
> gdalmove is a Python utility, and is only available if GDAL Python bindings are available.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-s_srs <srs_defn>

Override the coordinate system of the file with the indicated coordinate system definition. Optional. If not provided the source coordinate system is read from the source file.

-t_srs <srs_defn>

Defines the target coordinate system. This coordinate system will be written to the file after an update.

-et <max_pixel_err>

The error threshold (in pixels) beyond which the file will not be updated. If not provided no update will be applied to the file, but errors will be reported.

<target_file>

The file to be operated on. To update this must be a file format that supports in place updates of the geotransform and SRS.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

5.2.2.31 gdalsrsinfo

Synopsis

```
Usage: gdalsrsinfo [--help] [--help-general]
                   [--single-line] [-V] [-e][-o <out_type>] <srs_def>
```

Description

The **gdalsrsinfo** utility reports information about a given SRS from one of the following:

- The filename of a dataset supported by GDAL/OGR which contains SRS information
- Any of the usual GDAL/OGR forms (complete WKT, PROJ.4, EPSG:n or a file containing the SRS)

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

--single-line

Print WKT on single line

-V

Validate SRS

-e

Search for EPSG number(s) corresponding to SRS

-o <out_type>

Output types:

- `default`: proj4 and wkt (default option)
- `all`: all options available
- `wkt_all`: all wkt options available
- `PROJJSON`: PROJJSON string (GDAL >= 3.1 and PROJ >= 6.2)
- `proj4`: PROJ.4 string
- `wkt1`: OGC WKT format (full)
- `wkt_simple`: OGC WKT 1 (simplified)
- `wkt_noct`: OGC WKT 1 (without OGC CT params)
- `wkt_esri`: ESRI WKT format
- `wkt`: Latest WKT version supported, currently wkt2_2019
- `wkt2`: Latest WKT2 version supported, currently wkt2_2019
- `wkt2_2015`: OGC WKT2:2015
- `wkt2_2019`: OGC WKT2:2019 (for GDAL < 3.6, use `wkt2_2018`)

- `mapinfo`: Mapinfo style CoordSys format
- `xml`: XML format (GML based)

<srs_def>

may be the filename of a dataset supported by GDAL/OGR from which to extract SRS information OR any of the usual GDAL/OGR forms (complete WKT, PROJ.4, EPSG:n or a file containing the SRS)

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Example

Example 1

Default output

```
$ gdalsrsinfo EPSG:4326

PROJ.4 : +proj=longlat +datum=WGS84 +no_defs

OGC WKT2:2019 :
GEOGCRS["WGS 84",
    ENSEMBLE["World Geodetic System 1984 ensemble",
        MEMBER["World Geodetic System 1984 (Transit)"],
        MEMBER["World Geodetic System 1984 (G730)"],
        MEMBER["World Geodetic System 1984 (G873)"],
        MEMBER["World Geodetic System 1984 (G1150)"],
        MEMBER["World Geodetic System 1984 (G1674)"],
        MEMBER["World Geodetic System 1984 (G1762)"],
        MEMBER["World Geodetic System 1984 (G2139)"],
        ELLIPSOID["WGS 84",6378137,298.257223563,
            LENGTHUNIT["metre",1]],
        ENSEMBLEACCURACY[2.0]],
    PRIMEM["Greenwich",0,
        ANGLEUNIT["degree",0.0174532925199433]],
    CS[ellipsoidal,2],
        AXIS["geodetic latitude (Lat)",north,
            ORDER[1],
            ANGLEUNIT["degree",0.0174532925199433]],
        AXIS["geodetic longitude (Lon)",east,
            ORDER[2],
            ANGLEUNIT["degree",0.0174532925199433]],
    USAGE[
        SCOPE["Horizontal component of 3D system."],
        AREA["World."],
        BBOX[-90,-180,90,180]],
    ID["EPSG",4326]]
```

Example 2: PROJ.4 output

```
$ gdalsrsinfo -o proj4 lcc_esri.prj

+proj=lcc +lat_0=33.75 +lon_0=-79 +lat_1=36.1666666666667 +lat_2=34.3333333333333 +x_
↪0=609601.22 +y_0=0 +datum=NAD83 +units=m +no_defs
```

```
$ gdalsrsinfo -o proj4 landsat.tif
PROJ.4 : '+proj=utm +zone=19 +south +datum=WGS84 +units=m +no_defs '
```

Example 3: WKT output, latest version

```
$ gdalsrsinfo  -o wkt "EPSG:32722"

PROJCRS["WGS 84 / UTM zone 22S",
    BASEGEOGCRS["WGS 84",
        ENSEMBLE["World Geodetic System 1984 ensemble",
            MEMBER["World Geodetic System 1984 (Transit)"],
            MEMBER["World Geodetic System 1984 (G730)"],
            MEMBER["World Geodetic System 1984 (G873)"],
            MEMBER["World Geodetic System 1984 (G1150)"],
            MEMBER["World Geodetic System 1984 (G1674)"],
            MEMBER["World Geodetic System 1984 (G1762)"],
            MEMBER["World Geodetic System 1984 (G2139)"],
            ELLIPSOID["WGS 84",6378137,298.257223563,
                LENGTHUNIT["metre",1]],
            ENSEMBLEACCURACY[2.0]],
        PRIMEM["Greenwich",0,
            ANGLEUNIT["degree",0.0174532925199433]],
        ID["EPSG",4326]],
    CONVERSION["UTM zone 22S",
        METHOD["Transverse Mercator",
            ID["EPSG",9807]],
        PARAMETER["Latitude of natural origin",0,
            ANGLEUNIT["degree",0.0174532925199433],
            ID["EPSG",8801]],
        PARAMETER["Longitude of natural origin",-51,
            ANGLEUNIT["degree",0.0174532925199433],
            ID["EPSG",8802]],
        PARAMETER["Scale factor at natural origin",0.9996,
            SCALEUNIT["unity",1],
            ID["EPSG",8805]],
        PARAMETER["False easting",500000,
            LENGTHUNIT["metre",1],
            ID["EPSG",8806]],
        PARAMETER["False northing",10000000,
            LENGTHUNIT["metre",1],
            ID["EPSG",8807]]],
    CS[Cartesian,2],
        AXIS["(E)",east,
            ORDER[1],
            LENGTHUNIT["metre",1]],
```

(continues on next page)

(continued from previous page)

```
        AXIS["(N)",north,
            ORDER[2],
            LENGTHUNIT["metre",1]],
    USAGE[
        SCOPE["Navigation and medium accuracy spatial referencing."],
        AREA["Between 54°W and 48°W, southern hemisphere between 80°S and equator,␣
↪onshore and offshore. Brazil. Uruguay."],
        BBOX[-80,-54,0,-48]],
    ID["EPSG",32722]]
```

Example 4: WKT output, all versions

```
$ gdalsrsinfo -o wkt_all "EPSG:4322"

OGC WKT1 :
GEOGCS["WGS 72",
    DATUM["World_Geodetic_System_1972",
        SPHEROID["WGS 72",6378135,298.26,
            AUTHORITY["EPSG","7043"]],
        AUTHORITY["EPSG","6322"]],
    PRIMEM["Greenwich",0,
        AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.0174532925199433,
        AUTHORITY["EPSG","9122"]],
    AXIS["Latitude",NORTH],
    AXIS["Longitude",EAST],
    AUTHORITY["EPSG","4322"]]

OGC WKT2:2015 :
GEODCRS["WGS 72",
    DATUM["World Geodetic System 1972",
        ELLIPSOID["WGS 72",6378135,298.26,
            LENGTHUNIT["metre",1]]],
    PRIMEM["Greenwich",0,
        ANGLEUNIT["degree",0.0174532925199433]],
    CS[ellipsoidal,2],
        AXIS["geodetic latitude (Lat)",north,
            ORDER[1],
            ANGLEUNIT["degree",0.0174532925199433]],
        AXIS["geodetic longitude (Lon)",east,
            ORDER[2],
            ANGLEUNIT["degree",0.0174532925199433]],
    SCOPE["Horizontal component of 3D system."],
    AREA["World."],
    BBOX[-90,-180,90,180],
    ID["EPSG",4322]]

OGC WKT2:2019 :
GEOGCRS["WGS 72",
    DYNAMIC[
        FRAMEEPOCH[1972]],
```

(continues on next page)

(continued from previous page)

```
    DATUM["World Geodetic System 1972",
        ELLIPSOID["WGS 72",6378135,298.26,
            LENGTHUNIT["metre",1]]],
    PRIMEM["Greenwich",0,
        ANGLEUNIT["degree",0.0174532925199433]],
    CS[ellipsoidal,2],
        AXIS["geodetic latitude (Lat)",north,
            ORDER[1],
            ANGLEUNIT["degree",0.0174532925199433]],
        AXIS["geodetic longitude (Lon)",east,
            ORDER[2],
            ANGLEUNIT["degree",0.0174532925199433]],
    USAGE[
        SCOPE["Horizontal component of 3D system."],
        AREA["World."],
        BBOX[-90,-180,90,180]],
    ID["EPSG",4322]]

OGC WKT1 (simple) :
GEOGCS["WGS 72",
    DATUM["World_Geodetic_System_1972",
        SPHEROID["WGS 72",6378135,298.26]],
    PRIMEM["Greenwich",0],
    UNIT["degree",0.0174532925199433]]

OGC WKT1 (no CT) :
GEOGCS["WGS 72",
    DATUM["World_Geodetic_System_1972",
        SPHEROID["WGS 72",6378135,298.26]],
    PRIMEM["Greenwich",0],
    UNIT["degree",0.0174532925199433]]

ESRI WKT :
GEOGCS["GCS_WGS_1972",
    DATUM["D_WGS_1972",
        SPHEROID["WGS_1972",6378135.0,298.26]],
    PRIMEM["Greenwich",0.0],
    UNIT["Degree",0.0174532925199433]]
```

5.2.2.32 gdaltindex

Synopsis

```
Usage: gdaltindex [--help] [--long-usage] [--help-general]
                  [-overwrite] [-recursive] [-filename_filter <val>]...
                  [-min_pixel_size <val>] [-max_pixel_size <val>]
                  [-of <output_format>] [-tileindex <field_name>]
                  [-write_absolute_path] [-skip_different_projection]
                  [-t_srs <srs_def>] [-src_srs_name <field_name>]
                  [-src_srs_format {AUTO|WKT|EPSG|PROJ}] [-lyr_name <name>]
                  [-lco <NAME>=<VALUE>]... [-gti_filename <filename>]
                  [-tr <xres> <yres>] [-te <xmin> <ymin> <xmax> <ymax>]
```

(continues on next page)

(continued from previous page)

```
                    [-ot <datatype>] [-bandcount <val>] [-nodata <val>]...
                    [-colorinterp <val>]... [-mask] [-mo <name>=<value>]...
                    [-fetch_md <gdal_md_name> <fld_name> <fld_type>]... [--quiet]
                    <index_file> <file_or_dir>...
```

Description

This program creates an OGR-supported dataset with a record for each input raster file, an attribute containing the filename, and a polygon geometry outlining the raster. This output is suitable for use with MapServer as a raster tileindex, or as input for the *GTI* driver.

Tip

Equivalent in new "gdal" command line interface:

See *gdal raster index*.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-q, --quiet

Suppress progress messages and other non-error output.

-overwrite

Added in version 3.9.

Overwrite the tile index if it already exists.

-recursive

Added in version 3.9.

Whether directories specified in <file_or_dir> should be explored recursively.

-filename_filter <val>

Added in version 3.9.

Pattern that the filenames contained in directories pointed by <file_or_dir> should follow. '*' is a wildcard character that matches any number of any characters including none. '?' is a wildcard character that matches a single character. Comparisons are done in a case insensitive way. Several filters may be specified.

For example :`-filename_filter "*.tif" -filename_filter "*.tiff"`

-min_pixel_size <val>

Added in version 3.9.

Minimum pixel size in term of geospatial extent per pixel (resolution) that a raster should have to be selected. The pixel size is evaluated after reprojection of its extent to the target SRS defined by *-t_srs*.

-max_pixel_size <val>

Added in version 3.9.

Maximum pixel size in term of geospatial extent per pixel (resolution) that a raster should have to be selected. The pixel size is evaluated after reprojection of its extent to the target SRS defined by *-t_srs*.

-of <output_format>

The OGR format of the output tile index file. If not specified, the format is guessed from the extension.

-tileindex <field_name>

The output field name to hold the file path/location to the indexed rasters. The default tile index field name is `location`.

-write_absolute_path

The absolute path to the raster files is stored in the tile index file. By default the raster filenames will be put in the file exactly as they are specified on the command line.

-skip_different_projection

Only files with same projection as files already inserted in the tileindex will be inserted (unless *-t_srs* is specified). Default does not check projection and accepts all inputs.

-t_srs <target_srs>

Geometries of input files will be transformed to the desired target coordinate reference system. Default creates simple rectangular polygons in the same coordinate reference system as the input rasters.

-src_srs_name <field_name>

The name of the field to store the SRS of each tile. This field name can be used as the value of the TILESRS keyword in MapServer

-src_srs_format {AUTO|WKT|EPSG|PROJ}

The format in which the SRS of each tile must be written. Types can be `AUTO`, `WKT`, `EPSG`, `PROJ`. This option should be used together with *-src_srs_format*.

-lyr_name <name>

Layer name to create/append to in the output tile index file.

-lco <NAME>=<VALUE>

Added in version 3.9.

Layer creation option (format specific)

<index_file>

The name of the output file to create/append to. The default dataset will be created if it doesn't already exist, otherwise it will append to the existing dataset.

<file_or_dir>

The input GDAL raster files, can be multiple files separated by spaces. Wildcards may also be used. Stores the file locations in the same style as specified here, unless *-write_absolute_path* option is also used.

Starting with GDAL 3.9, this can also be a directory name. *-recursive* can also be used to recurse down to sub-directories.

It is also possible to use the generic option `--optfile filelist.txt` to specify a list of source files.

Options specific to use by the GDAL GTI driver

gdaltindex can be used to generate a tile index suitable for use by the *GTI* driver. There are two possibilities:

- either use directly a vector tile index generated by gdaltindex as the input of the GTI driver
- or generate a small XML .gti wrapper file, for easier use with non-file-based formats such as databases, or for vector formats that do not support setting layer metadata items.

Formats that support layer metadata are for example GeoPackage (`-of GPKG`), FlatGeoBuf (`-of FlatGeoBuf`) or PostGIS (`-of PG`)

Setting *-tr* and *-ot* is recommended to avoid the GTI driver to have to deduce them by opening the first tile in the index. If the tiles have nodata or mask band, *-nodata* and *-mask* should also be set.

In a GTI context, the extent of all tiles referenced in the tile index must be expressed in a single SRS. Consequently, if input tiles may have different SRS, either *-t_srs* or *-skip_different_projection* should be specified.

-gti_filename <name>

Added in version 3.9.

Filename of the XML Virtual Tile Index file to generate, that can be used as an input for the GDAL GTI / Virtual Raster Tile Index driver. This can be useful when writing the tile index in a vector format that does not support writing layer metadata items.

-tr <xres> <yres>

Added in version 3.9.

Target resolution in SRS unit per pixel.

Written in the XML Virtual Tile Index if *-gti_filename* is specified, or as RESX and RESY layer metadata items for formats that support layer metadata.

-te <xmin> <ymin> <xmax> <ymax>

Added in version 3.9.

Target extent in SRS unit.

Written in the XML Virtual Tile Index if *-gti_filename* is specified, or as MINX, MINY, MAXX and MAXY layer metadata items for formats that support layer metadata.

-ot <datatype>

Added in version 3.9.

Data type of the tiles of the tile index: Byte, Int8, UInt16, Int16, UInt32, Int32, UInt64, Int64, Float32, Float64, CInt16, CInt32, CFloat32 or CFloat64

Written in the XML Virtual Tile Index if *-gti_filename* is specified, or as DATA_TYPE layer metadata item for formats that support layer metadata.

-bandcount <val>

Added in version 3.9.

Number of bands of the tiles of the tile index.

Written in the XML Virtual Tile Index if *-gti_filename* is specified, or as BAND_COUNT layer metadata item for formats that support layer metadata.

A mix of tiles with N and N+1 bands is allowed, provided that the color interpretation of the (N+1)th band is alpha. The N+1 value must be written as the band count in that situation.

If *-nodata* or *-colorinterp* are specified and have multiple values, the band count is also inferred from that number.

-nodata <val>[,<val>...]

Added in version 3.9.

Nodata value of the tiles of the tile index.

Written in the XML Virtual Tile Index if *-gti_filename* is specified, or as NODATA layer metadata item for formats that support layer metadata.

-colorinterp <val>[,<val>...]

Added in version 3.9.

Color interpretation of of the tiles of the tile index: `red`, `green`, `blue`, `alpha`, `gray`, `undefined`.

Written in the XML Virtual Tile Index if *-gti_filename* is specified, or as `COLOR_INTERPRETATION` layer metadata item for formats that support layer metadata.

-mask

Added in version 3.9.

Whether tiles in the tile index have a mask band.

Written in the XML Virtual Tile Index if *-gti_filename* is specified, or as `MASK_BAND` layer metadata item for formats that support layer metadata.

-mo <KEY>=<VALUE>

Added in version 3.9.

Write an arbitrary layer metadata item, for formats that support layer metadata. This option may be repeated.

> **Note**
>
> This option cannot be used together *-gti_filename*

-fetch_md <gdal_md_name> <fld_name> <fld_type>

Added in version 3.9.

Fetch a metadata item from the raster tile and write it as a field in the tile index.

<gdal_md_name> should be the name of the raster metadata item. `{PIXEL_SIZE}` may be used as a special name to indicate the pixel size.

<fld_name> should be the name of the field to create in the tile index.

<fld_type> should be the name of the type to create. One of `String`, `Integer`, `Integer64`, `Real`, `Date`, `DateTime`

This option may be repeated.

For example: `-fetch_md TIFFTAG_DATETIME creation_date DateTime`

Examples

Example 1

Produce a shapefile (`doq_index.shp`) with a record for every image that the utility found in the `doq` folder. Each record holds information that points to the location of the image and also a bounding rectangle shape showing the bounds of the image:

```
gdaltindex doq_index.shp doq/*.tif
```

Example 2

Perform the same command as before, but now we create a GeoPackage instead of a Shapefile.

```
gdaltindex -of GPKG doq_index.gpkg doq/*.tif
```

Example 3

The *-t_srs* option can also be used to transform all input rasters into the same output projection:

```
gdaltindex -t_srs EPSG:4326 -src_srs_name src_srs tile_index_mixed_srs.shp *.tif
```

Example 4

Make a tile index from files listed in a text file, with metadata suitable for use by the GDAL GTI / Virtual Raster Tile Index driver.

```
gdaltindex tile_index.gti.gpkg -ot Byte -tr 60 60 -colorinterp Red,Green,Blue --optfile␣
↪my_list.txt
```

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

C API

This utility is also callable from C with *GDALTileIndex()*.

See also

Common options for raster programs for other command-line options, and in particular the *--optfile* switch that can be used to specify a list of input datasets.

5.2.2.33 gdaltransform

Synopsis

```
Usage: gdaltransform [--help] [--help-general]
       [-i] [-s_srs <srs_def>] [-t_srs <srs_def>] [-to >NAME>=<VALUE>]...
       [-s_coord_epoch <epoch>] [-t_coord_epoch <epoch>]
       [-ct <proj_string>] [-order <n>] [-tps] [-rpc] [-geoloc]
       [-gcp <pixel> <line> <easting> <northing> [elevation]]...
       [-output_xy] [-E] [-field_sep <sep>] [-ignore_extra_input]
       [<srcfile> [<dstfile>]]
```

Description

The gdaltransform utility reprojects a list of coordinates into any supported projection,including GCP-based transformations.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-s_srs <srs_def>

Set source spatial reference. The coordinate systems that can be passed are anything supported by the OGRSpatialReference.SetFromUserInput() call, which includes EPSG PCS and GCSes (i.e. EPSG:4296), PROJ.4 declarations (as above), or the name of a .prj file containing well known text.

-s_coord_epoch <epoch>

Added in version 3.8.

Assign a coordinate epoch, linked with the source SRS. Useful when the source SRS is a dynamic CRS. Only taken into account if *-s_srs* is used.

Before PROJ 9.4, *-s_coord_epoch* and *-t_coord_epoch* were mutually exclusive, due to lack of support for transformations between two dynamic CRS.

-t_srs <srs_def>

set target spatial reference. The coordinate systems that can be passed are anything supported by the OGRSpatialReference.SetFromUserInput() call, which includes EPSG PCS and GCSes (i.e. EPSG:4296), PROJ.4 declarations (as above), or the name of a .prj file containing well known text.

-t_coord_epoch <epoch>

Added in version 3.8.

Assign a coordinate epoch, linked with the output SRS. Useful when the output SRS is a dynamic CRS. Only taken into account if *-t_srs* is used.

Before PROJ 9.4, *-s_coord_epoch* and *-t_coord_epoch* were mutually exclusive, due to lack of support for transformations between two dynamic CRS.

-ct <string>

A PROJ string (single step operation or multiple step string starting with +proj=pipeline), a WKT2 string describing a CoordinateOperation, or a urn:ogc:def:coordinateOperation:EPSG::XXXX URN overriding the default transformation from the source to the target CRS. It must take into account the axis order of the source and target CRS.

Added in version 3.0.

-to <NAME>=<VALUE>

set a transformer option suitable to pass to *GDALCreateGenImgProjTransformer2()*.

-order <n>

order of polynomial used for warping (1 to 3). The default is to select a polynomial order based on the number of GCPs.

-tps

Force use of thin plate spline transformer based on available GCPs.

-rpc

Force use of RPCs.

-geoloc

Force use of Geolocation Arrays.

-i

Inverse transformation: from destination to source.

-gcp <pixel> <line> <easting> <northing> [<elevation>]

Provide a GCP to be used for transformation (generally three or more are required). Pixel and line need not be integers.

-output_xy

Restrict output to "x y" instead of "x y z"

-ignore_extra_input

Added in version 3.9.

Set this flag to avoid extra non-numeric content at end of input lines to be appended to the output lines.

-E

Added in version 3.9.

Enable Echo mode, where input coordinates are prepended to the output lines.

-field_sep <sep>

Added in version 3.9.

Defines the field separator, to separate different values. It defaults to the space character.

<srcfile>

Raster dataset with source projection definition or GCPs. If not given, source projection/GCPs are read from the command-line *-s_srs* or *-gcp* parameters.

Note that only the SRS and/or GCPs of this input file is taken into account, and not its pixel content.

<dstfile>

Raster dataset with destination projection definition.

Coordinates are read as pairs, triples (for 3D,) or (since GDAL 3.0.0,) quadruplets (for X,Y,Z,time) of numbers per line from standard input, transformed, and written out to standard output in the same way. All transformations offered by gdalwarp are handled, including gcp-based ones.

Starting with GDAL 3.9, additional non-numeric content (typically point name) at the end of an input line will also be appended to the output line, unless the *-ignore_extra_input* is added.

Note that input and output must always be in decimal form. There is currently no support for DMS input or output.

If an input image file is provided, input is in pixel/line coordinates on that image. If an output file is provided, output is in pixel/line coordinates on that image.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Reprojection Example

Simple reprojection from one projected coordinate system to another:

```
gdaltransform -s_srs EPSG:28992 -t_srs EPSG:31370
177502 311865
```

Produces the following output in meters in the "Belge 1972 / Belgian Lambert 72" projection:

```
244296.724777415 165937.350217148 0
```

Image RPC Example

The following command requests an RPC based transformation using the RPC model associated with the named file. Because the -i (inverse) flag is used, the transformation is from output georeferenced (WGS84) coordinates back to image coordinates.

```
gdaltransform -i -rpc 06OCT20025052-P2AS-005553965230_01_P001.TIF
125.67206 39.85307 50
```

Produces this output measured in pixels and lines on the image:

```
3499.49282422381 2910.83892848414 50
```

X,Y,Z,time transform

15-term time-dependent Helmert coordinate transformation from ITRF2000 to ITRF93 for a coordinate at epoch 2000.0

```
gdaltransform -ct "+proj=pipeline +step +proj=unitconvert +xy_in=deg \
+xy_out=rad +step +proj=cart +step +proj=helmert +convention=position_vector \
+x=0.0127 +dx=-0.0029 +rx=-0.00039 +drx=-0.00011 +y=0.0065 +dy=-0.0002 \
+ry=0.00080 +dry=-0.00019 +z=-0.0209 +dz=-0.0006 +rz=-0.00114 +drz=0.00007 \
+s=0.00195 +ds=0.00001 +t_epoch=1988.0 +step +proj=cart +inv +step \
+proj=unitconvert +xy_in=rad +xy_out=deg"
2 49 0 2000
```

Produces this output measured in longitude degrees, latitude degrees and ellipsoid height in meters:

```
2.0000005420366 49.0000003766711 -0.0222802283242345
```

Ground control points

Task: find one address and assign another. We pick Salt Lake City, where road names *are* their grid values. We first establish some ground control points at road intersections. We'll use *--optfile* for easy reuse of our GCPs.

```
echo -output_xy \
-gcp 0   0    -111.89114717 40.76932606 \
-gcp 0   -500 -111.89114803 40.75846686 \
-gcp 500 0    -111.87685039 40.76940631 > optfile.txt
```

Where is the address "370 S. 300 E."?

```
echo 300 -370 Address 370 S. 300 E. | gdaltransform --optfile optfile.txt
-111.8825697384 40.761338402 Address 370 S. 300 E.
```

(The 370 is protected from being consumed as an elevation by placing a word before it.)

Nearby, a newly constructed building needs an address assigned. We use *-i*:

```
echo -111.88705 40.76502 Building ABC123 | gdaltransform -i --optfile optfile.txt
143.301947786644 -199.32683635161 Building ABC123
```

(i.e., 143 E. 200 S. Or 144 if across the street.)

5.2.2.34 gdalwarp

Synopsis

```
Usage: gdalwarp [--help] [--long-usage] [--help-general]
                [--quiet] [-overwrite] [-of <output_format>]
```

(continues on next page)

(continued from previous page)

```
                [-co <NAME>=<VALUE>]... [-s_srs <srs_def>] [-t_srs <srs_def>]
                [[-srcalpha]|[-nosrcalpha]]
                [-dstalpha] [-tr <xres> <yres>|square] [-ts <width> <height>]
                [-te <xmin> <ymin> <xmax> <ymax>] [-te_srs <srs_def>]
                [-r␣
↪near|bilinear|cubic|cubicspline|lanczos|average|rms|mode|min|max|med|q1|q3|sum]
                [-ot Byte|Int8|[U]Int{16|32|64}|CInt{16|32}|[C]Float{32|64}]
                <src_dataset_name>... <dst_dataset_name>

Advanced options:
                [-wo <NAME>=<VALUE>]... [-multi] [-s_coord_epoch <epoch>]
                [-t_coord_epoch <epoch>] [-ct <string>]
                [[-tps]|[-rpc]|[-geoloc]]
                [-order <1|2|3>] [-refine_gcps <tolerance> [<minimum_gcps>]]
                [-to <NAME>=<VALUE>]... [-et <err_threshold>]
                [-wm <memory_in_mb>] [-srcnodata "<value>[ <value>]..."]
                [-dstnodata "<value>[ <value>]..."] [-tap]
                [-wt Byte|Int8|[U]Int{16|32|64}|CInt{16|32}|[C]Float{32|64}]
                [-cutline <datasource>|<WKT>] [-cutline_srs <srs_def>]
                [-cwhere <expression>]
                [[-cl <layername>]|[-csql <query>]]
                [-cblend <distance>] [-crop_to_cutline] [-nomd]
                [-cvmd <meta_conflict_value>] [-setci] [-oo <NAME>=<VALUE>]...
                [-doo <NAME>=<VALUE>]... [-ovr <level>|AUTO|AUTO-<n>|NONE]
                [[-vshift]|[-novshiftgrid]]
                [-if <format>]... [-srcband <band>]... [-dstband <band>]...
```

Description

The **gdalwarp** utility is an image mosaicing, reprojection and warping utility. The program can reproject to any supported projection, and can also apply GCPs stored with the image if the image is "raw" with control information.

> **Tip**
>
> Equivalent in new "gdal" command line interface:
>
> - *gdal raster reproject* for reprojection
> - *gdal raster update* to update the content of a raster with another one

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-q, --quiet

Suppress progress messages and other non-error output.

-b <n>

-srcband <n>

Added in version 3.7.

Specify an input band number to warp (between 1 and the number of bands of the source dataset).

This option is used to warp a subset of the input bands. All input bands are used when it is not specified.

This option may be repeated multiple times to select several input bands. The order in which bands are specified will be the order in which they appear in the output dataset (unless *-dstband* is specified).

The alpha band should not be specified in the list, as it will be automatically retrieved (unless *-nosrcalpha* is specified).

The following invocation will warp an input datasets with bands ordered as Blue, Green, Red, NearInfraRed in an output dataset with bands ordered as Red, Green, Blue.

```
gdalwarp in_bgrn.tif out_rgb.tif -b 3 -b 2 -b 1 -overwrite
```

-dstband <n>

Added in version 3.7.

Specify the output band number in which to warp. In practice, this option is only useful when updating an existing dataset, e.g to warp one band at at time.

```
gdal_create -if in_red.tif -bands 3 out_rgb.tif
gdalwarp in_red.tif out_rgb.tif -srcband 1 -dstband 1
gdalwarp in_green.tif out_rgb.tif -srcband 1 -dstband 2
gdalwarp in_blue.tif out_rgb.tif -srcband 1 -dstband 3
```

If *-srcband* is specified, there must be as many occurrences of *-dstband* as there are of *-srcband*.

The output alpha band should not be specified, as it will be automatically created if the input dataset has an alpha band, or if *-dstalpha* is specified.

If *-dstband* is not specified, then `-dstband 1 -dstband 2 ... -dstband N` is assumed where N is the number of input bands (specified explicitly either with *-srcband* or implicitly)

-s_srs <srs def>

Set source spatial reference. If not specified the SRS found in the input dataset will be used.

The coordinate reference systems that can be passed are anything supported by the OGRSpatialReference.SetFromUserInput() call, which includes EPSG Projected, Geographic or Compound CRS (i.e. EPSG:4296), a well known text (WKT) CRS definition, PROJ.4 declarations, or the name of a .prj file containing a WKT CRS definition.

If the SRS has an explicit vertical datum that points to a geoid grid, and the input dataset is a single band dataset, a vertical correction will be applied to the values of the dataset.

-s_coord_epoch <epoch>

Added in version 3.4.

Assign a coordinate epoch, linked with the source SRS. Useful when the source SRS is a dynamic CRS. Only taken into account if *-s_srs* is used.

Before PROJ 9.4, *-s_coord_epoch* and *-t_coord_epoch* were mutually exclusive, due to lack of support for transformations between two dynamic CRS.

-t_srs <srs_def>

Set target spatial reference.

A source SRS must be available for reprojection to occur. The source SRS will be by default the one found in the input dataset when it is available, or as overridden by the user with *-s_srs*

The coordinate reference systems that can be passed are anything supported by the OGRSpatialReference.SetFromUserInput() call, which includes EPSG Projected, Geographic or Compound CRS (i.e.

EPSG:4296), a well known text (WKT) CRS definition, PROJ.4 declarations, or the name of a .prj file containing a WKT CRS definition.

If the SRS has an explicit vertical datum that points to a geoid grid, and the input dataset is a single band dataset, a vertical correction will be applied to the values of the dataset.

-t_coord_epoch <epoch>

Added in version 3.4.

Assign a coordinate epoch, linked with the target SRS. Useful when the target SRS is a dynamic CRS. Only taken into account if *-t_srs* is used.

Before PROJ 9.4, *-s_coord_epoch* and *-t_coord_epoch* were mutually exclusive, due to lack of support for transformations between two dynamic CRS.

-ct <string>

A PROJ string (single step operation or multiple step string starting with +proj=pipeline), a WKT2 string describing a CoordinateOperation, or a urn:ogc:def:coordinateOperation:EPSG::XXXX URN overriding the default transformation from the source to the target CRS.

It must take into account the axis order of the source and target CRS, that is typically include a `step proj=axisswap order=2,1` at the beginning of the pipeline if the source CRS has northing/easting axis order, and/or at the end of the pipeline if the target CRS has northing/easting axis order.

When creating a new output file, using *-t_srs* is still necessary to have the target CRS written in the metadata of the output file, but the parameters of the CoordinateOperation will override those of the standard transformation.

Added in version 3.0.

-to <NAME>=<VALUE>

Set a transformer option suitable to pass to *GDALCreateGenImgProjTransformer2()*. See *GDALCreateRPCTransformerV2()* for RPC specific options.

-vshift

Force the use of vertical shift. This option is generally not necessary, except when using an explicit coordinate transformation (*-ct*), and not specifying an explicit source and target SRS.

Added in version 3.4.

-novshift

Disable the use of vertical shift when one of the source or target SRS has an explicit vertical datum, and the input dataset is a single band dataset.

> **Note**
>
> this option was named `-novshiftgrid` until GDAL 3.3.

Added in version 3.4.

-order <n>

order of polynomial used for warping (1 to 3). The default is to select a polynomial order based on the number of GCPs.

-tps

Force use of Thin Plate Spline transformer based on available GCPs.

> **Warning**
>
> Using a Thin Plate Spline transformer with more than a few thousand GCPs requires significant RAM usage (at least `numGCPs` * `numGCPs` * 8 bytes) and processing time.

-rpc

Force use of RPCs.

-geoloc

Force use of Geolocation Arrays.

-et `<err_threshold>`

Error threshold for transformation approximation, expressed as a number of source pixels. Defaults to 0.125 pixels unless the `RPC_DEM` transformer option is specified, in which case an exact transformer, i.e. `err_threshold=0`, will be used.

-refine_gcps `<tolerance> [<minimum_gcps>]`

Refines the GCPs by automatically eliminating outliers. Outliers will be eliminated until minimum_gcps are left or when no outliers can be detected. The tolerance is passed to adjust when a GCP will be eliminated. Note that GCP refinement only works with polynomial interpolation. The tolerance is in pixel units if no projection is available, otherwise it is in SRS units. If minimum_gcps is not provided, the minimum GCPs according to the polynomial model is used.

-te `<xmin> <ymin> <xmax> <ymax>`

Set georeferenced extents of output file to be created (in target SRS by default, or in the SRS specified with *-te_srs*)

-te_srs `<srs_def>`

Specifies the SRS in which to interpret the coordinates given with -te. The <srs_def> may be any of the usual GDAL/OGR forms, complete WKT, PROJ.4, EPSG:n or a file containing the WKT. This must not be confused with -t_srs which is the target SRS of the output dataset. *-te_srs* is a convenience e.g. when knowing the output coordinates in a geodetic long/lat SRS, but still wanting a result in a projected coordinate system.

-tr `<xres> <yres> | -tr square`

Set output file resolution (in target georeferenced units).

If not specified (or not deduced from -te and -ts), gdalwarp will, in the general case, generate an output raster with xres=yres.

Starting with GDAL 3.7, if neither *-tr* nor *-ts* are specified, that no reprojection is involved (including taking into account geolocation arrays or RPC), the resolution of the source file(s) will be preserved (in previous version, an output raster with xres=yres was always generated). It is possible to ask square pixels to still be generated, by specifying `square` as the value for *-tr*.

-tap

(target aligned pixels) align the coordinates of the extent of the output file to the values of the *-tr*, such that the aligned extent includes the minimum extent (edges lines/columns that are detected as blank, before actual warping, will be removed starting with GDAL 3.8). Alignment means that xmin / resx, ymin / resy, xmax / resx and ymax / resy are integer values. It does not necessarily mean that the output grid aligns with the input grid.

-ts `<width> <height>`

Set output file size in pixels and lines. If width or height is set to 0, the other dimension will be guessed from the computed resolution. Note that *-ts* cannot be used with *-tr*

-ovr <level>|AUTO|AUTO-<n>|NONE

To specify which overview level of source files must be used. The default choice, AUTO, will select the overview level whose resolution is the closest to the target resolution. Specify an integer value (0-based, i.e. 0=1st overview level) to select a particular level. Specify AUTO-n where n is an integer greater or equal to 1, to select an overview level below the AUTO one. Or specify NONE to force the base resolution to be used (can be useful if overviews have been generated with a low quality resampling method, and the warping is done using a higher quality resampling method).

-wo <NAME>=<VALUE>

Set a warp option. The *GDALWarpOptions::papszWarpOptions* docs show all options. Multiple *-wo* options may be listed.

-ot <type>

Force the output image bands to have a specific data type supported by the driver, which may be one of the following: `Byte`, `Int8`, `UInt16`, `Int16`, `UInt32`, `Int32`, `UInt64`, `Int64`, `Float32`, `Float64`, `CInt16`, `CInt32`, `CFloat32` or `CFloat64`.

-wt <type>

Working pixel data type. The data type of pixels in the source image and destination image buffers.

-r <resampling_method>

Resampling method to use. Available methods are:

`near`: nearest neighbour resampling (default for non-COG drivers, fastest algorithm, worst interpolation quality).

`bilinear`: bilinear resampling.

`cubic`: cubic resampling (default for COG driver).

`cubicspline`: cubic spline resampling.

`lanczos`: Lanczos windowed sinc resampling.

`average`: average resampling, computes the weighted average of all non-NODATA contributing pixels.

`rms` root mean square / quadratic mean of all non-NODATA contributing pixels (GDAL >= 3.3)

`mode`: mode resampling, selects the value which appears most often of all the sampled points. In the case of ties, the first value identified as the mode will be selected.

`max`: maximum resampling, selects the maximum value from all non-NODATA contributing pixels.

`min`: minimum resampling, selects the minimum value from all non-NODATA contributing pixels.

`med`: median resampling, selects the median value of all non-NODATA contributing pixels.

`q1`: first quartile resampling, selects the first quartile value of all non-NODATA contributing pixels.

`q3`: third quartile resampling, selects the third quartile value of all non-NODATA contributing pixels.

`sum`: compute the weighted sum of all non-NODATA contributing pixels (since GDAL 3.1)

> **Note**
>
> When downsampling is performed (use of *-tr* or *-ts*), existing overviews (either internal/implicit or external ones) on the source image will be used by default by selecting the closest overview to the desired output resolution. The resampling method used to create those overviews is generally not the one you specify through the *-r* option. Some formats, like JPEG2000, can contain significant outliers due to how wavelet compression works. It might thus be useful in those situations to use the *-ovr* `NONE` option to prevent existing overviews to be used.

-srcnodata "<value>[<value>]..."

Set nodata masking values for input bands (different values can be supplied for each band). If more than one value is supplied all values should be quoted to keep them together as a single operating system argument. Masked values will not be used in interpolation (details given in *Nodata / source validity mask handling*)

Use a value of `None` to ignore intrinsic nodata settings on the source dataset.

When this option is set to a non-`None` value, it causes the `UNIFIED_SRC_NODATA` warping option (see *GDALWarpOptions::papszWarpOptions*) to be set to `YES`, if it is not explicitly set.

If `-srcnodata` is not explicitly set, but the source dataset has nodata values, they will be taken into account, with `UNIFIED_SRC_NODATA` at `PARTIAL` by default.

-dstnodata "<value>[<value>]..."

Set nodata values for output bands (different values can be supplied for each band). If more than one value is supplied all values should be quoted to keep them together as a single operating system argument. New files will be initialized to this value and if possible the nodata value will be recorded in the output file. Use a value of `None` to ensure that nodata is not defined. If this argument is not used then nodata values will be copied from the source dataset.

-srcalpha

Force the last band of a source image to be considered as a source alpha band.

-nosrcalpha

Prevent the alpha band of a source image to be considered as such (it will be warped as a regular band)

-dstalpha

Create an output alpha band to identify nodata (unset/transparent) pixels.

-wm <memory_in_mb>

Set the amount of memory that the warp API is allowed to use for caching. Defaults to 64 MB. Since GDAL 3.10, the value can be specified either as a fixed amount of memory (e.g., `-wm 200MB`, `-wm 1G`) or as a percentage of usable RAM (`-wm 10%`). In earlier versions, or if a unit is not specified, the value is interpreted as being in megabytes if the value is less than 10000. For values >=10000, it is interpreted as bytes.

The warper will total up the memory required to hold the input and output image arrays and any auxiliary masking arrays and if they are larger than the "warp memory" allowed it will subdivide the chunk into smaller chunks and try again.

If the -wm value is very small there is some extra overhead in doing many small chunks so setting it larger is better but it is a matter of diminishing returns.

-multi

Use multithreaded warping implementation. Two threads will be used to process chunks of image and perform input/output operation simultaneously. Note that computation is not multithreaded itself. To do that, you can use the *-wo* NUM_THREADS=val/ALL_CPUS option, which can be combined with *-multi*

-if <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

Added in version 3.2.

-of <format>

Select the output format. If not specified, the format is guessed from the extension. Use the short format name.

-co <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created. For instance, the GeoTIFF driver supports creation options to control compression, and whether the file should be tiled.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--format* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

-cutline <datasource>|<WKT>

Enable use of a blend cutline from the name of a vector dataset. Starting with GDAL 3.9, a WKT geometry string starting with POLYGON or MULTIPOLYGON can also be specified.

-cutline_srs <srs_def>

Added in version 3.9.

Sets or overrides the SRS of the cutline.

-cl <layername>

Select the named layer from the cutline datasource.

-cwhere <expression>

Restrict desired cutline features based on attribute query.

-csql <query>

Select cutline features using an SQL query instead of from a layer with *-cl*.

-cblend <distance>

Set a blend distance to use to blend over cutlines (in pixels).

-crop_to_cutline

Crop the extent of the target dataset to the extent of the cutline.

-overwrite

Overwrite the target dataset if it already exists. Overwriting must be understood here as deleting and recreating the file from scratch. Note that if this option is *not* specified and the output file already exists, it will be updated in place.

-nomd

Do not copy metadata. Without this option, dataset and band metadata (as well as some band information) will be copied from the first source dataset. Items that differ between source datasets will be set to * (see *-cvmd* option).

-cvmd <meta_conflict_value>

Value to set metadata items that conflict between source datasets (default is "*"). Use "" to remove conflicting items.

-setci

Set the color interpretation of the bands of the target dataset from the source dataset.

-oo <NAME>=<VALUE>

Dataset open option (format specific)

-doo <NAME>=<VALUE>

Output dataset open option (format specific)

<src_dataset_name>

The source file name(s).

<dst_dataset_name>

The destination file name.

Overview

gdalwarp transforms images between different coordinate reference systems and spatial resolutions.

First, **gdalwarp** must determine the extent and resolution of the output, if these have not been specified using *-te* and *-tr*. These are determined by transforming a sample of points from the source CRS to the destination CRS. Details of the procedure can be found in the documentation for *GDALSuggestedWarpOutput()*. If multiple inputs are provided to **gdalwarp**, the output extent will be calculated to cover all of them, at a resolution consistent with the highest-resolution input.

Once the dimensions of the output image have been determined, **gdalwarp** divides the output image into chunks that can be processed independently within the amount of memory specified by *-wm*. **gdalwarp** then iterates over scanlines in these chunks, and for each output pixel determines a rectangular region of source pixels that contribute to the value of the output pixel. The dimensions of this rectangular region are typically determined by estimating the relative scales of the source and destination raster, but can be manually specified (see documentation of the `XSCALE` parameter in *GDALWarpOptions::papszWarpOptions*). Because the source region is a simple rectangle, it is not possible for an output pixel to be associated with source pixels from both sides of the antimeridian or pole (when transforming from geographic coordinates).

The rectangular region of source pixels is then provided to a function that performs the resampling algorithm selected with *-r*. Depending on the resampling algorithm and relative scales of the source and destination rasters, source pixels may be weighted either according to the approximate fraction of the source pixel that is covered by the destination pixel (e.g., "mean" and "sum" resampling), or by horizontal and vertical Cartesian distances between the center of the source pixel and the center of the target pixel (e.g., bilinear or cubic spline resampling). In the latter case, the relative weight of an individual source pixel is determined by the product of the weights determined for its row and column; the diagonal Cartesian distance is not calculated.

Multiple input files

When multiple inputs are provided to **gdalwarp**, they are processed independently in the order they are listed. This may introduce edge effects near the boundaries of the input files, because output pixel values will be derived from the final input only. To avoid this, non-overlapping input files may first be combined into a *VRT file* (e.g., using *gdalbuildvrt*). This will allow **gdalwarp** to use pixels from all inputs when calculating output pixel values.

Writing to an existing file

Mosaicing into an existing output file is supported if the output file already exists. The spatial extent of the existing file will not be modified to accommodate new data, so you may have to remove it in that case, or use the *-overwrite* option.

Polygon cutlines may be used as a mask to restrict the area of the destination file that may be updated, including blending. If the OGR layer containing the cutline features has no explicit SRS, the cutline features are assumed to be in the SRS of the destination file. When writing to a not yet existing target dataset, its extent will be the one of the original raster unless *-te* or *-crop_to_cutline* are specified.

Nodata / source validity mask handling

Invalid values in source pixels, either identified through a nodata value metadata set on the source band, a mask band, an alpha band or the use of `-srcnodata` will not be used in interpolation. The details of how it is taken into account depends on the resampling kernel:

- for nearest resampling, for each target pixel, the coordinate of its center is projected back to source coordinates and the source pixel containing that coordinate is identified. If this source pixel is invalid, the target pixel is considered as nodata.
- for bilinear, cubic, cubicspline and lanczos, for each target pixel, the coordinate of its center is projected back to source coordinates and a corresponding source pixel is identified. If this source pixel is invalid, the target pixel is considered as nodata (in this case, valid pixels within the kernel radius would not be considered). Given that those resampling kernels have a non-null kernel radius, this source pixel is just one among other several source pixels, and it might be possible that there are invalid values in those other contributing source pixels. The weights used to take into account those invalid values will be set to zero to ignore them.
- for the other resampling methods, source pixels contributing to the target pixel are ignored if invalid. Only the valid ones are taken into account. If there are none, the target pixel is considered as nodata.

If using `-srcnodata` for multiple images with different invalid values, you need to either (a) pre-process them to have the same to-be-ignored value, or (b) set the nodata flag for each file. Use (b) if you need to preserve the original values for some reason, for example:

```
# for this image we want to ignore black (0)
gdalwarp -srcnodata 0 -dstnodata 0 orig-ignore-black.tif black-nodata.tif

# and now we want to ignore white (0)
gdalwarp -srcnodata 255 -dstnodata 255 orig-ignore-white.tif white-nodata.tif

# and finally ignore a particular blue-grey (RGB 125 125 150)
gdalwarp -srcnodata "125 125 150" -dstnodata "125 125 150" orig-ignore-grey.tif grey-
↪nodata.tif

# now we can mosaic them all and not worry about nodata parameters
gdalwarp black-nodata.tif grey-nodata.tif white-nodata.tif final-mosaic.tif
```

Approximate transformation

By default **gdalwarp** uses a linear approximator for the transformations with a permitted error of 0.125 pixels in the source dataset. For each processing chunk, the approximator precisely transforms three points per output scanline (the start, middle, and end) from a row and column in the output dataset to a row and column in the source dataset. It then compares a linear approximation of the center point coordinates to the precisely transformed value. If the sum of the horizontal and vertical errors is less than the error threshold then the remaining source points are approximated using linear interpolation between the start and middle point, and between the middle and end point. If the error exceeds the threshold, the scanline is split into two sections and the approximator is recursively applied to each section until the error is less than the threshold or all points have been exactly computed.

Note that a processing chunk does not necessarily correspond to a whole destination scanline. If the output dataset is tiled, its shape will typically be one or several tiles. The `OPTIMIZE_SIZE` warping option and the value of the `-wm` option can also influence the shape of the processing chunk. Consequently, for a single (non-zero) value of the error threshold, the linear approximation might result in slightly different values in coordinate transformation, but all of them within the permitted error.

The error threshold (in source dataset pixels) can be controlled with the gdalwarp `-et` switch. If you want to compare a true pixel-by-pixel reprojection use `-et 0` which disables this approximator entirely.

Vertical transformation

While gdalwarp is most commonly used to perform coordinate transformations in the 2D space, it can also perform vertical transformations. Vertical transformations are automatically performed when the following two conditions are met:

- at least one of the source or target CRS has an explicit vertical CRS (as part of a compound CRS) or is a 3D (generally geographic) CRS, and
- the raster has a single band

This mode can also be forced by using the *-vshift* (this is essentially useful when the CRS involved are not explicitly 3D, but a transformation pipeline is specified with *-ct*), or disabled with *-novshift*.

When a vertical transformation is involved, typically a shift value read in a geoid grid will be applied. This may require such grid(s) to be installed, or PROJ networking capabilities to be enabled. Consult PROJ documentation for more details. In addition to a shift, the raster values may be multiplied by a factor to take into account vertical unit changes. In priority, the value returned by *GDALRasterBand::GetUnitType()* is used. The following values are currently recognized: `m`, `metre`, `metre`, `ft`, `foot`, `US survey foot`. If there is no defined unit type at the band level, the vertical unit of the source CRS is used. The vertical unit of the target CRS is also used to determine that conversion factor. The conversion factor may be overridden by setting the `MULT_FACTOR_VERTICAL_SHIFT` warping option with *-wo*. For example `-wo MULT_FACTOR_VERTICAL_SHIFT=1` to disable any vertical unit change.

Memory usage

Adding RAM will almost certainly increase the speed of **gdalwarp**. That's not at all the same as saying that it is worth it, or that the speed increase will be significant. Disks are the slowest part of the process. By default **gdalwarp** won't take much advantage of RAM. Using the flag *-wm 500* will operate on 500MB chunks at a time which is better than the default. The warp memory specified by *-wm* is shared among all threads, so it is especially beneficial to increase this value when running **gdalwarp** with *-wo NUM_THREADS* (or its equivalent *GDAL_NUM_THREADS*) greater than 1.

Increasing the I/O block cache size may also help. This can be done by setting the *GDAL_CACHEMAX* configuration like:

```
gdalwarp --config GDAL_CACHEMAX 500 -wm 500 ...
```

This uses 500MB of RAM for read/write caching, and 500MB of RAM for working buffers during the warp. Beyond that it is doubtful more memory will make a substantial difference.

Check CPU usage while **gdalwarp** is running. If it is substantially less than 100% then you know things are IO bound. Otherwise they are CPU bound. The `--debug` option may also provide useful information. For instance, after running the following:

```
gdalwarp --debug on abc.tif def.tif
```

a message like the following will be output:

```
GDAL: 224 block reads on 32 block band 1 of utm.tif
```

In this case it is saying that band 1 of `utm.tif` has 32 blocks, but that 224 block reads were done, implying that lots of data was having to be re-read, presumably because of a limited IO cache. You will also see messages like:

```
GDAL: GDALWarpKernel()::GWKNearestNoMasksByte()
Src=0,0,512x512 Dst=0,0,512x512
```

The Src/Dst windows show you the "chunk size" being used. In this case my whole image which is very small. If you find things are being broken into a lot of chunks increasing *-wm* may help somewhat.

But far more important than memory are ensuring you are going through an optimized path in the warper. If you ever see it reporting `GDALWarpKernel()::GWKGeneralCase()` you know things will be relatively slow. Basically, the fastest situations are nearest neighbour resampling on 8bit data without nodata or alpha masking in effect.

Compressed output

In some cases, the output of **gdalwarp** may be much larger than the original, even if the same compression algorithm is used. By default, **gdalwarp** operates on chunks that are not necessarily aligned with the boundaries of the blocks/tiles/strips of the output format, so this might cause repeated compression/decompression of partial blocks, leading to lost space in the output format.

The situation can be improved by using the `OPTIMIZE_SIZE` warping option (*-wo OPTIMIZE_SIZE=YES*), but note that depending on the source and target projections, it might also significantly slow down the warping process.

Another possibility is to use **gdalwarp** without compression and then follow up with **gdal_translate** with compression:

```
gdalwarp infile tempfile.tif ...options...
gdal_translate tempfile.tif outfile.tif -co compress=lzw ...etc.
```

Alternatively, you can use a VRT file as the output format of **gdalwarp**. The VRT file is just an XML file that will be created immediately. The **gdal_translate** operations will be of course a bit slower as it will do the real warping operation.

```
gdalwarp -of VRT infile tempfile.vrt ...options...
gdal_translate tempfile.vrt outfile.tif -co compress=lzw ...etc.
```

Frequently Asked Questions

Q1. Why does the quality of the output looks so bad (no anti-aliasing)?

Did you specify a resampling method, with *-r*, other than the default nearest neighbour?

Q2. Why do I get slightly different results whether the output dataset is tiled or not?

This is related to the fact that an approximate coordinate transformation is used by default to speed-up computation. If you want to get the same results whether the output is tiled or not, set *-et* to zero. Note, however, that this will only work for relatively small images; other factors can still result in different result. See following question (Q3).

Q3. Why do I observe artifacts, that look like resolution changes and are aligned with rectangular areas of the output raster, when warping sufficiently large rasters, particularly in areas where the reprojection involves significant deformation and only with non-nearest resampling ?

The warping engine operates on rectangular areas of the output dataset (generally aligned with tile boundaries for a compressed tile dataset).

During reprojection, a single source pixel does not generally correspond to a single output pixel. The resampling method must therefore properly account for this and compute a ratio between the number of source and target pixels in the horizontal (X) and vertical (Y) directions. These ratios are computed per warping chunk. This maximizes the local quality of the warping but has the downside of creating visual discontinuities between warping chunks.

If you favor a seamless result, you may manually specify the XSCALE and YSCALE warping options with *-wo*. The XSCALE (resp. YSCALE) value is the ratio expressing the resampling factor, i.e. the number of destination pixels per source pixel, along the horizontal (resp. vertical) axis. It equals to one for no resampling, is below one for downsampling, and above one for upsampling.

Examples

- Basic transformation:

```
gdalwarp -t_srs EPSG:4326 input.tif output.tif
```

- For instance, an eight bit spot scene stored in GeoTIFF with control points mapping the corners to lat/long could be warped to a UTM projection with a command like this:

```
gdalwarp -t_srs '+proj=utm +zone=11 +datum=WGS84' -overwrite raw_spot.tif utm11.tif
```

- For instance, the second channel of an ASTER image stored in HDF with control points mapping the corners to lat/long could be warped to a UTM projection with a command like this:

```
gdalwarp -overwrite HDF4_SDS:ASTER_L1B:"pg-PR1B0000-2002031402_100_001":2 \
    pg-PR1B0000-2002031402_100_001_2.tif
```

- To apply a cutline on a un-georeferenced image and clip from pixel (220,60) to pixel (1160,690):

```
gdalwarp -overwrite -to SRC_METHOD=NO_GEOTRANSFORM -to DST_METHOD=NO_GEOTRANSFORM \
    -te 220 60 1160 690 -cutline cutline.csv in.png out.tif
```

where cutline.csv content is like:

```
id,WKT
1,"POLYGON((....))"
```

- To transform a DEM from geoid elevations (using EGM96) to WGS84 ellipsoidal heights:

```
gdalwarp -overwrite in_dem.tif out_dem.tif -s_srs EPSG:4326+5773 -t_srs EPSG:4979
```

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

C API

This utility is also callable from C with *GDALWarp()*.

5.2.2.35 nearblack

Synopsis

```
Usage: nearblack [--help] [--long-usage] [--help-general]
                 [-of <output_format>] [--quiet] [-co <NAME>=<VALUE>]...
                 [-o <output_file>]
                 [[-white]|[-color <c1,c2,c3...cn>]...]
                 [-nb <non_black_pixels>] [-near <dist>] [-setalpha] [-setmask]
                 [-alg floodfill|twopasses]
                 <input_file>
```

Description

This utility will scan an image and try to set all pixels that are nearly or exactly black, white or one or more custom colors around the collar to black or white. This is often used to "fix up" lossy compressed air photos so that color pixels can be treated as transparent when mosaicing. The output format must use lossless compression if either alpha band or mask band is not set.

> **Tip**
>
> Equivalent in new "gdal" command line interface:
>
> See *gdal raster clean-collar*.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-q, --quiet

Suppress progress messages and other non-error output.

-o <outfile>

The name of the output file to be created.

-of <format>

Select the output format. If not specified, the format is guessed from the extension. Use the short format name (GTiff for GeoTIFF for example).

-co <NAME>=<VALUE>

Passes a creation option to the output format driver. Multiple *-co* options may be listed. See *Raster drivers* format specific documentation for legal creation options for each format.

Only valid when creating a new file

-white

Search for nearly white (255) pixels instead of nearly black pixels.

-color <c1>,<c2>,<c3>...<cn>

Search for pixels near the specified color. May be specified multiple times. When -color is specified, the pixels that are considered as the collar are set to 0.

-near <dist>

Select how far from black, white or custom colors the pixel values can be and still considered near black, white or custom color. Defaults to 15.

-nb <non_black_pixels>

number of consecutive non-black pixels that can be encountered before the giving up search inwards. Defaults to 2.

-setalpha

Adds an alpha band if the output file is specified and the input file has 3 bands, or sets the alpha band of the output file if it is specified and the input file has 4 bands, or sets the alpha band of the input file if it has 4 bands and no output file is specified. The alpha band is set to 0 in the image collar and to 255 elsewhere.

-setmask

Adds a mask band to the output file, or adds a mask band to the input file if it does not already have one and no output file is specified. The mask band is set to 0 in the image collar and to 255 elsewhere.

-alg `twopasses|floodfill`

Added in version 3.8.

Selects the algorithm to apply.

`twopasses` uses a top-to-bottom pass followed by a bottom-to-top pass. This is the only algorithm implemented before GDAL 3.8. It may miss with concave areas. The algorithm processes the image one scanline at a time. A scan "in" is done from either end setting pixels to black or white until at least "non_black_pixels" pixels that are more than "dist" gray levels away from black, white or custom colors have been encountered at which point the scan stops. The nearly black, white or custom color pixels are set to black or white. The algorithm also scans from top to bottom and from bottom to top to identify indentations in the top or bottom.

`floodfill` (added in GDAL 3.8) uses the Flood Fill algorithm and will work with concave areas. It requires creating a temporary dataset and is slower than `twopasses`. When a non-zero value for *-nb* is used, `twopasses` is actually called as an initial step of `floodfill`.

<infile>

The input file. Any GDAL supported format, any number of bands, normally 8bit Byte bands.

The processing is all done in 8bit (Bytes).

If the output file is omitted, the processed results will be written back to the input file - which must support update.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

C API

This utility is also callable from C with *GDALNearblack()*.

5.2.2.36 pct2rgb

Synopsis

```
pct2rgb [--help] [--help-general]
           [-of format] [-b band] [-rgba] [-pct palette_file] <source_file> <dest_file>
```

Description

This utility will convert a pseudo-color band on the input file into an output RGB file of the desired format.

> **Note**
>
> pct2rgb is a Python utility, and is only available if GDAL Python bindings are available.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-of <format>

Select the output format. If not specified, the format is guessed from the extension. Use the short format name.

-b <band>

Band to convert to RGB, defaults to 1.

-rgba

Generate a RGBA file (instead of a RGB file by default).

-pct <palette_file>

Extract the color table from <palette_file> instead of getting it from <source_file>. Can be used to have a consistent color table for multiple files. The <palette_file> must be either a raster file in a GDAL supported format with a palette or a color file in a supported format (txt, qml, qlr).

<source_file>

The input file.

<dest_file>

The output RGB file that will be created.

The '-expand rgb|rgba' option of *gdal_translate* obsoletes that utility.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

5.2.2.37 rgb2pct

Synopsis

```
rgb2pct [--help] [--help-general] [--creation-option OPTION]
           [-n colors | -pct palette_file] [-of format] <source_file> <dest_file>
```

Description

This utility will compute an optimal pseudo-color table for a given RGB image using a median cut algorithm on a downsampled RGB histogram. Then it converts the image into a pseudo-colored image using the color table. This conversion utilizes Floyd-Steinberg dithering (error diffusion) to maximize output image visual quality.

Note

rgb2pct is a Python utility, and is only available if GDAL Python bindings are available.

Tip

Equivalent in new "gdal" command line interface:

See *gdal raster rgb-to-palette*.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-n <color>

Select the number of colors in the generated color table. Defaults to 256. Must be between 2 and 256.

--creation-option OPTION

Optional creation parameters for the GeoTIFF driver, for example "COMPRESS=LZW". Can be specified multiple times.

-pct <palette_file>

Extract the color table from <palette_file> instead of computing it. Can be used to have a consistent color table for multiple files. The <palette_file> must be either a raster file in a GDAL supported format with a palette or a color file in a supported format (txt, qml, qlr).

-of <format>

Select the output format. If not specified, the format is guessed from the extension. Use the short format name. Only output formats supporting pseudo-color tables should be used.

<source_file>

The input RGB file.

<dest_file>

The output pseudo-colored file that will be created.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Example

If it is desired to hand create the palette, likely the simplest text format is the GDAL VRT format. In the following example a VRT was created in a text editor with a small 4 color palette with the RGBA colors 238/238/238/255, 237/237/237/255, 236/236/236/255 and 229/229/229/255.

```
$ rgb2pct -pct palette.vrt rgb.tif pseudo-colored.tif
$ more < palette.vrt
<VRTDataset rasterXSize="226" rasterYSize="271">
    <VRTRasterBand dataType="Byte" band="1">
        <ColorInterp>Palette</ColorInterp>
        <ColorTable>
        <Entry c1="238" c2="238" c3="238" c4="255"/>
        <Entry c1="237" c2="237" c3="237" c4="255"/>
        <Entry c1="236" c2="236" c3="236" c4="255"/>
        <Entry c1="229" c2="229" c3="229" c4="255"/>
        </ColorTable>
    </VRTRasterBand>
</VRTDataset>
```

5.2.3 Multidimensional Raster programs

5.2.3.1 gdalmdiminfo

Synopsis

```
Usage: gdalmdiminfo [--help] [--long-usage] [--help-general]
                    [[-summary]|[-detailed]]
                    [-nopretty] [-array <array_name>] [-limit <number>]
                    [-oo <NAME>=<VALUE>]... [-if <format>]...
                    [-arrayoption <NAME>=<VALUE>]... [-stats]
                    <dataset_name>
```

Description

gdalmdiminfo program lists various information about a GDAL supported multidimensional raster dataset as JSON output. It follows the following JSON schema gdalmdiminfo_output.schema.json.

> **Tip**
>
> Equivalent in new "gdal" command line interface:
>
> See *gdal mdim info*.

The following command line parameters can appear in any order

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-detailed

Most verbose output. Report attribute data types and array values.

-nopretty

Outputs on a single line without any indentation.

-array <array_name>

Name of the array used to restrict the output to the specified array.

-limit <number>

Number of values in each dimension that is used to limit the display of array values. By default, unlimited. Only taken into account if used with -detailed.

-oo <NAME>=<VALUE>

Dataset open option (format specific). This option may be used several times.

-arrayoption <NAME>=<VALUE>

Option passed to *GDALGroup::GetMDArrayNames()* to filter reported arrays. Such option is format specific. Consult driver documentation. This option may be used several times.

-stats

Read and display array statistics. Force computation if no statistics are stored in an array.

Added in version 3.2.

-if <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

Added in version 3.2.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

C API

This utility is also callable from C with *GDALMultiDimInfo()*.

Examples

Example 1: Display general structure

```
$ gdalmdiminfo netcdf-4d.nc
```

```
{
  "type": "group",
  "name": "/",
  "attributes": {
    "Conventions": "CF-1.5"
  },
  "dimensions": [
    {
      "name": "levelist",
      "full_name": "/levelist",
      "size": 2,
      "type": "VERTICAL",
      "indexing_variable": "/levelist"
    },
    {
      "name": "longitude",
      "full_name": "/longitude",
      "size": 10,
      "type": "HORIZONTAL_X",
      "direction": "EAST",
      "indexing_variable": "/longitude"
    },
    {
      "name": "latitude",
      "full_name": "/latitude",
      "size": 10,
      "type": "HORIZONTAL_Y",
      "direction": "NORTH",
      "indexing_variable": "/latitude"
    },
```

(continues on next page)

(continued from previous page)

```
    {
      "name": "time",
      "full_name": "/time",
        "size": 4,
      "type": "TEMPORAL",
      "indexing_variable": "/time"
      }
  ],
  "arrays": {
    "levelist": {
      "datatype": "Int32",
      "dimensions": [
          "/levelist"
        ],
      "attributes": {
        "long_name": "pressure_level"
      },
      "unit": "millibars"
    },
    "longitude": {
      "datatype": "Float32",
      "dimensions": [
        "/longitude"
      ],
      "attributes": {
        "standard_name": "longitude",
        "long_name": "longitude",
        "axis": "X"
      },
      "unit": "degrees_east"
    },
    "latitude": {
      "datatype": "Float32",
      "dimensions": [
        "/latitude"
      ],
      "attributes": {
        "standard_name": "latitude",
        "long_name": "latitude",
        "axis": "Y"
      },
      "unit": "degrees_north"
    },
    "time": {
      "datatype": "Float64",
      "dimensions": [
        "/time"
      ],
      "attributes": {
        "standard_name": "time",
        "calendar": "standard"
      },
```

(continues on next page)

(continued from previous page)

```
      "unit": "hours since 1900-01-01 00:00:00"
    },
    "t": {
      "datatype": "Int32",
      "dimensions": [
        "/time",
        "/levelist",
        "/latitude",
        "/longitude"
      ],
      "nodata_value": -32767
    }
  },
  "structural_info": {
    "NC_FORMAT": "CLASSIC"
  }
}
```

Example 2: Display detailed information about a given array

```
$ gdalmdiminfo netcdf-4d.nc -array t -detailed -limit 3
```

5.2.3.2 gdalmdimtranslate

Synopsis

```
Usage: gdalmdimtranslate [--help] [--long-usage] [--help-general]
                         [-if <format>]... [-of <output_format>]
                         [-co <NAME>=<VALUE>]...
                         [[-array <array_spec>]...]
                         [-arrayoption <NAME>=<VALUE>]...
                         [[-group <group_spec>]...]
                         [-subset <subset_spec>]...
                         [-scaleaxes <scaleaxes_spec>] [-strict]
                         [-oo <NAME>=<VALUE>]... [--quiet]
                         <src_dataset> <dst_dataset>
```

Description

gdalmdimtranslate program converts multidimensional raster between different formats, and/or can perform selective conversion of specific arrays and groups, and/or subsetting operations.

> **Tip**
>
> Equivalent in new "gdal" command line interface:
>
> See *gdal mdim convert*.

The following command line parameters can appear in any order.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-q, --quiet

Suppress progress messages and other non-error output.

-if <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

Added in version 3.2.

-of <format>

Select the output format. This can be a format that supports multidimensional output (such as *NetCDF: Network Common Data Form*, *Multidimensional VRT*), or a "classic" 2D formats, if only one single 2D array results of the other specified conversion operations. When this option is not specified, the format is guessed when possible from the extension of the destination filename.

-co <NAME>=<VALUE>

Many formats have one or more optional creation options that can be used to control particulars about the file created.

The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the *--format* command line option but the documentation for the format is the definitive source of information on driver creation options. See *Raster drivers* format specific documentation for legal creation options for each format.

Array-level creation options may be passed by prefixing them with `ARRAY:`. See *`GDALGroup::CopyFrom()`* for further details regarding such options.

-array <array_spec>

Instead of converting the whole dataset, select one array, and possibly perform operations on it. This option can be specified several times to operate on different arrays.

<array_spec> may be just an array name, potentially using a fully qualified syntax (/group/subgroup/array_name). Or it can be a combination of options with the syntax: name={src_array_name}[,dstname={dst_array_name}][,resample=yes][,transpose=[{axis1},{axis2},...][,view={view_expr}]

The following options are processed in that order:

- `resample=yes` asks for the array to run through *`GDALMDArray::GetResampled()`*.
- **[{axis1},{axis2},...] is the argument of *`GDALMDArray::Transpose()`*.**
  For example, transpose=[1,0] switches the axis order of a 2D array. See *Example 5*.
- {view_expr} is the value of the *viewExpr* argument of *`GDALMDArray::GetView()`*. See *Example 4*.

When specifying a view_expr that performs a slicing or subsetting on a dimension, the equivalent operation will be applied to the corresponding indexing variable.

-arrayoption <NAME>=<VALUE>

Added in version 3.9.

Option passed to *`GDALGroup::GetMDArrayNames()`* to filter reported arrays. Such option is format specific. Consult driver documentation. This option may be used several times.

-group <group_spec>

Instead of converting the whole dataset, select one group, and possibly perform operations on it. This option can be specified several times to operate on different groups. If only one group is specified, its content will be copied directly to the target root group. If several ones are specified, they are copied under the target root group

<group_spec> may be just a group name, potentially using a fully qualified syntax (/group/subgroup/subsubgroup_name). Or it can be a combination of options with the syntax: name={src_group_name}[,dstname={dst_group_name}][,recursive=no]

-subset <subset_spec>

Performs a subsetting (trimming or slicing) operation along a dimension, provided that it is indexed by a 1D variable of numeric or string data type, and whose values are monotonically sorted. <subset_spec> follows exactly the OGC WCS 2.0 KVP encoding for subsetting.

That is dim_name(min_val,max_val) or dim_name(sliced_val) The first syntax will subset the dimension dim_name to values in the [min_val,max_val] range. The second syntax will slice the dimension dim_name to value sliced_val (and this dimension will be removed from the arrays that reference to it)

Using -subset is incompatible with specifying a *view* option in -array.

See *Example 2*.

-scaleaxes <scaleaxes_spec>

Applies a integral scale factor to one or several dimensions, that is extract 1 value every N values (without resampling).

<scaleaxes_spec> follows exactly the syntax of the KVP encoding of the SCALEAXES parameter of OGC WCS 2.0 Scaling Extension, but limited to integer scale factors.

That is <dim1_name>(<scale_factor>)[,<dim2_name>(<scale_factor>)]...

Using -scaleaxes is incompatible with specifying a *view* option in -array.

See *Example 3*.

-strict

By default, some failures during the translation are tolerated, such as not being able to write group attributes. When setting this option, such failures will cause the process to fail.

-oo <NAME>=<VALUE>

Added in version 3.4.

Source dataset open option (format specific)

<src_dataset>

The source dataset name.

<dst_dataset>

The destination file name.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

C API

This utility is also callable from C with *GDALMultiDimTranslate()*.

Examples

Example 1: Convert a netCDF file to a multidimensional VRT file

```
gdalmdimtranslate in.nc out.vrt
```

Example 2: Extract a 2D slice of a time,Y,X array

```
gdalmdimtranslate in.nc out.tif -subset 'time("2010-01-01")' -array temperature
```

Example 3: Subsample along X and Y axis

```
gdalmdimtranslate in.nc out.nc -scaleaxes "X(2),Y(2)"
```

Example 4: Reorder the values of an array

Reorder the values of the time,Y,X array along the Y axis from top-to-bottom to bottom-to-top (or the reverse)

```
gdalmdimtranslate in.nc out.nc -array "name=temperature,view=[:,::-1,:]"
```

Example 5: Transpose an array that has X,Y,time dimension order to time,Y,X

```
gdalmdimtranslate in.nc out.nc -array "name=temperature,transpose=[2,1,0]"
```

5.2.4 Vector programs

5.2.4.1 Common options for vector programs

All GDAL OGR command line programs support the following common options.

--version

Report the version of GDAL and exit.

--build

Report detailed information about GDAL in use.

--formats

List all vector formats supported by this GDAL build (read-only and read-write) and exit. The format support is indicated as follows:

- `ro` is read-only driver
- `rw` is read or write (i.e. supports *GDALDriver::CreateCopy()*)
- `rw+` is read, write and update (i.e. supports *GDALDriver::Create()*)
- A `v` is appended for formats supporting virtual IO (`/vsimem`, `/vsigzip`, `/vsizip`, etc).
- A `s` is appended for formats supporting subdatasets.

The order in which drivers are listed is the one in which they are registered, which determines the order in which they are successively probed when opening a dataset. Most of the time, this order does not matter, but in some situations, several drivers may recognize the same file. The `-if` option of some utilities can be specified to restrict opening the dataset with a subset of drivers (generally one). Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions. Alternatively, the *GDAL_SKIP* configuration option can also be used to exclude one or several drivers.

--format <format>

List detailed information about a single format driver. The format should be the short name reported in the *--formats* list, such as GML.

--formats

List all drivers. Can be combined with `-json` switch of **ogrinfo** of since GDAL 3.10

--optfile <filename>

Read the named file and substitute the contents into the command line options list. Lines beginning with # will be ignored. Multi-word arguments may be kept together with double quotes.

--config <key> <value>

Sets the named configuration keyword to the given value, as opposed to setting them as environment variables. Some common configuration keywords are *SHAPE_ENCODING* (force shapefile driver to read DBF files with the given character encoding) and *CPL_TMPDIR* (define the location of temporary files). Individual drivers may be influenced by other *configuration options*.

--debug <value>

Control what debugging messages are emitted. A value of ON will enable all debug messages. A value of OFF will disable all debug messages. Otherwise, a debug message will be emitted if its category appears somewhere in the value string.

--help-general

Gives a brief usage message for the generic GDAL OGR command line options and exit.

5.2.4.2 ogrinfo

Synopsis

```
Usage: ogrinfo [--help] [--long-usage] [--help-general]
               [-json] [-ro] [-update] [--quiet] [-fid <FID>]
               [-spat <xmin> <ymin> <xmax> <ymax>] [-geomfield <field>]
               [-where <restricted_where>]
               [[-sql <statement|@filename>]|[-rl]]
               [-dialect <dialect>] [-al]
               [[-summary]|[-features]]
               [-limit <nb_features>] [-fields YES|NO]
               [-geom YES|NO|SUMMARY|WKT|ISO_WKT] [-oo <NAME=VALUE>]... [-nomd]
               [-listmdd] [-mdd <domain>]... [-nocount] [-noextent] [-extent3D]
               [-nogeomtype] [-wkt_format WKT1|WKT2|WKT2_2015|WKT2_2019]
               [-fielddomain <name>] [-if <format>]...
               filename [<layer_name>]...
```

Description

The **ogrinfo** program lists various information about an OGR-supported data source to stdout (the terminal). By executing SQL statements it is also possible to edit data.

Tip

Equivalent in new "gdal" command line interface:

See *gdal vector info*.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-if <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

Added in version 3.2.

-json

Display the output in json format, conforming to the ogrinfo_output.schema.json ogrinfo_output.schema.json schema.

Added in version 3.7.

-ro

Open the data source in read-only mode.

-al

List all layers (used instead of having to give layer names as arguments). In the default text output, this also enables listing all features, which can be disabled with *-so*. In JSON output, -al is implicit, but listing of features must be explicitly enabled with *-features*.

-rl

Enable random layer reading mode, i.e. iterate over features in the order they are found in the dataset, and not layer per layer. This can be significantly faster for some formats (for example OSM, GMLAS). -rl cannot be used with -sql.

-so

Summary Only: suppress listing of individual features and show only summary information like projection, schema, feature count and extents. In JSON output, -so is implicit and listing of features can be enabled with *-features*.

-features

Enable listing of features. This has the opposite effect of *-so*.

This option should be used with caution if using the library function *GDALVectorInfo()* and/or *-json*, as the whole output of ogrinfo will be built in memory. Consequently, when used on a large collection of features, RAM may be exhausted.

Added in version 3.7.

-limit <nb_features>

Added in version 3.9.

Limit the number of features per layer.

-q

Quiet verbose reporting of various information, including coordinate system, layer schema, extents, and feature count.

-where <restricted_where>

An attribute query in a restricted form of the queries used in the SQL *WHERE* statement. Only features matching the attribute query will be reported. The @<filename> syntax can be used to indicate that the content is in the pointed filename.

Example of -where and quoting:

```
-where "\"Corner Point Identifier\" LIKE '%__00_00'"
```

Note: -dialect is ignored with -where. Use -sql instead of -where if you want to use -dialect.

-sql <statement>|@<filename>

Execute the indicated SQL statement and return the result. The @<filename> syntax can be used to indicate that the content is in the pointed filename (e.g @my_select.txt where my_select.txt is a file in the current directory). Data can also be edited with SQL INSERT, UPDATE, DELETE, DROP TABLE, ALTER TABLE etc. Editing capabilities depend on the selected dialect with *-dialect*.

-dialect <dialect>

SQL dialect. In some cases can be used to use (unoptimized) *OGR SQL dialect* instead of the native SQL of an RDBMS by passing the OGRSQL dialect value. The *SQL SQLite dialect* can be selected with the SQLITE and INDIRECT_SQLITE dialect values, and this can be used with any datasource. Note: -dialect is ignored with -where. Use -sql instead of -where if you want to use -dialect.

-spat <xmin> <ymin> <xmax> <ymax>

The area of interest. Only features within the rectangle will be reported.

-geomfield <field>

Name of the geometry field on which the spatial filter operates.

-fid <fid>

If provided, only the feature with this feature id will be reported. Operates exclusive of the spatial or attribute queries. Note: if you want to select several features based on their feature id, you can also use the fact the 'fid' is a special field recognized by OGR SQL. So, *-where "fid in (1,3,5)"* would select features 1, 3 and 5.

-fields=YES|NO

If set to NO, the feature dump will not display field values. Default value is YES.

-fielddomain <domain_name>

Added in version 3.3.

Display details about a field domain.

-geom=YES|NO|SUMMARY|WKT|ISO_WKT

If set to NO, the feature dump will not display the geometry. If set to SUMMARY, only a summary of the geometry will be displayed. If set to YES or ISO_WKT, the geometry will be reported in full OGC WKT format. If set to WKT the geometry will be reported in legacy WKT. Default value is YES.

-oo <NAME>=<VALUE>

Dataset open option (format specific).

If a driver supports the OGR_SCHEMA open option, it can be used to partially or completely override the auto-detected schema (i.e. which fields are read, with which types, subtypes, length, precision etc.) of the dataset.

The value of this option is a JSON string or a path to a JSON file that complies with the OGR_SCHEMA open option schema definition

-nomd

Suppress metadata printing. Some datasets may contain a lot of metadata strings.

-listmdd

List all metadata domains available for the dataset.

-mdd <domain>

Report metadata for the specified domain. `all` can be used to report metadata in all domains.

-nocount

Suppress feature count printing.

-noextent

Suppress spatial extent printing.

-extent3D

Added in version 3.9.

Request a 3D extent to be reported (the default is 2D only). Note that this operation might be slower than requesting the 2D extent, depending on format and driver capabilities.

-nogeomtype

Suppress layer geometry type printing.

Added in version 3.1.

--formats

List all vector formats supported by this GDAL build (read-only and read-write) and exit. The format support is indicated as follows:

- `ro` is read-only driver
- `rw` is read or write (i.e. supports *GDALDriver::CreateCopy()*)
- `rw+` is read, write and update (i.e. supports *GDALDriver::Create()*)
- A `v` is appended for formats supporting virtual IO (`/vsimem`, `/vsigzip`, `/vsizip`, etc).
- A `s` is appended for formats supporting subdatasets.

The order in which drivers are listed is the one in which they are registered, which determines the order in which they are successively probed when opening a dataset. Most of the time, this order does not matter, but in some situations, several drivers may recognize the same file. The `-if` option of some utilities can be specified to restrict opening the dataset with a subset of drivers (generally one). Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions. Alternatively, the *GDAL_SKIP* configuration option can also be used to exclude one or several drivers.

-wkt_format <format>

The WKT format used to display the SRS. Currently supported values for the `format` are:

`WKT1`

`WKT2` (latest WKT version, currently *WKT2_2018*)

`WKT2_2015`

`WKT2_2018`

Added in version 3.0.0.

<datasource_name>

The data source to open. May be a filename, directory or other virtual name. See the OGR Vector Formats list for supported datasources.

<layer>

One or more layer names may be reported. If no layer names are passed then ogrinfo will report a list of available layers (and their layer wide geometry type). If layer name(s) are given then their extents, coordinate system, feature count, geometry type, schema and all features matching query parameters will be reported to the terminal. If no query parameters are provided, all features are reported.

Geometries are reported in OGC WKT format.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

C API

This utility is also callable from C with *GDALVectorInfo()*.

Added in version 3.7.

Examples

Example 1: Reporting the names of the layers in a NTF file

```
$ ogrinfo wrk/SHETLAND_ISLANDS.NTF

INFO: Open of `wrk/SHETLAND_ISLANDS.NTF'
using driver `UK .NTF' successful.
1: BL2000_LINK (Line String)
2: BL2000_POLY (None)
3: BL2000_COLLECTIONS (None)
4: FEATURE_CLASSES (None)
```

Example 2: Retrieving a summary (-so) of a layer without showing details about every feature

```
$ ogrinfo \
  -so \
  natural_earth_vector.gpkg \
  ne_10m_admin_0_antarctic_claim_limit_lines

INFO: Open of `natural_earth_vector.gpkg'
     using driver `GPKG' successful.

Layer name: ne_10m_admin_0_antarctic_claim_limit_lines
Geometry: Line String
Feature Count: 23
Extent: (-150.000000, -90.000000) - (160.100000, -60.000000)
Layer SRS WKT:
GEOGCS["WGS 84",
    DATUM["WGS_1984",
```

(continues on next page)

(continued from previous page)

```
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0,
        AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.0174532925199433,
        AUTHORITY["EPSG","9122"]],
    AUTHORITY["EPSG","4326"]]
FID Column = fid
Geometry Column = geom
type: String (15.0)
scalerank: Integer (0.0)
featurecla: String (50.0)
```

Example 3: Retrieving information on a file in JSON format without showing details about every feature

```
ogrinfo -json poly.shp
```


```
{
  "description":"autotest/ogr/data/poly.shp",
  "driverShortName":"ESRI Shapefile",
  "driverLongName":"ESRI Shapefile",
  "layers":[
    {
      "name":"poly",
      "metadata":{
        "":{
          "DBF_DATE_LAST_UPDATE":"2018-08-02"
        },
        "SHAPEFILE":{
          "SOURCE_ENCODING":""
        }
      },
      "geometryFields":[
        {
          "name":"",
          "type":"Polygon",
          "nullable":true,
          "extent":[
            478315.53125,
            4762880.5,
            481645.3125,
            4765610.5
          ],
          "coordinateSystem":{
            "wkt":"PROJCRS[\"OSGB36 / British National Grid\",BASEGEOGCRS[\"OSGB36\",
↪DATUM...",
            "projjson":{
              "$schema":"https://proj.org/schemas/v0.6/projjson.schema.json",
              "type":"ProjectedCRS",
```


(continues on next page)

(continued from previous page)

```
            "name":"OSGB36 / British National Grid",
            "base_crs":{
              "name":"OSGB36",
              "datum":{
                "type":"GeodeticReferenceFrame",
                "name":"Ordnance Survey of Great Britain 1936",
                "ellipsoid":{
                  "name":"Airy 1830",
                  "semi_major_axis":6377563.396,
                  "inverse_flattening":299.3249646
                }
              },
              "coordinate_system":{
                "subtype":"ellipsoidal",
                "axis":[
                  {
                    "name":"Geodetic latitude",
                    "abbreviation":"Lat",
                    "direction":"north",
                    "unit":"degree"
                  },
                  {
                    "name":"Geodetic longitude",
                    "abbreviation":"Lon",
                    "direction":"east",
                    "unit":"degree"
                  }
                ]
              },
              "id":{
                "authority":"EPSG",
                "code":4277
              }
            },
            "conversion":{
              "name":"British National Grid",
              "method":{
                "name":"Transverse Mercator",
                "id":{
                  "authority":"EPSG",
                  "code":9807
                }
              },
              "parameters":[
                {
                  "name":"Latitude of natural origin",
                  "value":49,
                  "unit":"degree",
                  "id":{
                    "authority":"EPSG",
                    "code":8801
                  }
```

(continues on next page)

(continued from previous page)

```
        },
        {
          "name":"Longitude of natural origin",
          "value":-2,
          "unit":"degree",
          "id":{
            "authority":"EPSG",
            "code":8802
          }
        },
        {
          "name":"Scale factor at natural origin",
          "value":0.9996012717,
          "unit":"unity",
          "id":{
            "authority":"EPSG",
            "code":8805
          }
        },
        {
          "name":"False easting",
          "value":400000,
          "unit":"metre",
          "id":{
            "authority":"EPSG",
            "code":8806
          }
        },
        {
          "name":"False northing",
          "value":-100000,
          "unit":"metre",
          "id":{
            "authority":"EPSG",
            "code":8807
          }
        }
      ]
    },
    "coordinate_system":{
      "subtype":"Cartesian",
      "axis":[
        {
          "name":"Easting",
          "abbreviation":"E",
          "direction":"east",
          "unit":"metre"
        },
        {
          "name":"Northing",
          "abbreviation":"N",
          "direction":"north",
```

(continues on next page)

(continued from previous page)


```
                "unit":"metre"
              }
            ]
          },
          "scope":"Engineering survey, topographic mapping.",
          "area":"United Kingdom (UK) - offshore to boundary of UKCS within 49°45...
↪",
          "bbox":{
            "south_latitude":49.75,
            "west_longitude":-9,
            "north_latitude":61.01,
            "east_longitude":2.01
          },
          "id":{
            "authority":"EPSG",
            "code":27700
          }
        },
        "dataAxisToSRSAxisMapping":[
          1,
          2
        ]
      }
    }
  ],
  "featureCount":10,
  "fields":[
    {
      "name":"AREA",
      "type":"Real",
      "width":12,
      "precision":3,
      "nullable":true,
      "uniqueConstraint":false
    },
    {
      "name":"EAS_ID",
      "type":"Integer64",
      "width":11,
      "nullable":true,
      "uniqueConstraint":false
    },
    {
      "name":"PRFEDEA",
      "type":"String",
      "width":16,
      "nullable":true,
      "uniqueConstraint":false
    }
  ]
}
],
```


(continues on next page)

(continued from previous page)

```
  "metadata":{
  },
  "domains":{
  },
  "relationships":{
  }
}
```

Example 4: Using -q and an attribute query to restrict the output to certain features in a layer

```
$ ogrinfo -q -ro \
    -where 'GLOBAL_LINK_ID=185878' \
    wrk/SHETLAND_ISLANDS.NTF BL2000_LINK

Layer name: BL2000_LINK
OGRFeature(BL2000_LINK):2
  LINE_ID (Integer) = 2
  GEOM_ID (Integer) = 2
  FEAT_CODE (String) = (null)
  GLOBAL_LINK_ID (Integer) = 185878
  TILE_REF (String) = SHETLAND I
  LINESTRING (419832.100 1069046.300,419820.100 1069043.800,...
```

Example 5: Updating a value of an attribute in a shapefile with SQL by using the SQLite dialect

```
ogrinfo test.shp -dialect sqlite -sql "update test set attr='bar' where attr='foo'"
```

Example 6: Adding a column to an input file

```
ogrinfo input.shp -sql "ALTER TABLE input ADD fieldX float"
```

Example 7: Performing a SQL query without an input file

Sometimes there is no input file involved in a calculation. In such cases one may use the `:memory:` input file which is a in-memory empty SQLite file (and the SQLite SQL dialect will be implicitly used).

```
ogrinfo :memory: -sql "SELECT ST_Buffer(ST_GeomFromText('POINT(0 0)'), 1)"
```

5.2.4.3 ogr2ogr

Synopsis

```
Usage: ogr2ogr [--help] [--long-usage] [--help-general]
               [-of <output_format>] [-dsco <NAME>=<VALUE>]...
               [-lco <NAME>=<VALUE>]...
               [[-append]|[-upsert]|[-overwrite]]
               [-update] [-sql <statement>|@<filename>] [-dialect <dialect>]
               [-spat <xmin> <ymin> <xmax> <ymax>]
               [-where <restricted_where>|@<filename>] [-select <field_list>]
```

(continues on next page)

(continued from previous page)

```
                [-nln <name>] [-nlt <type>]... [-s_srs <srs_def>]
                [[-a_srs <srs_def>]|[-t_srs <srs_def>]]
                <dst_dataset_name> <src_dataset_name> [<layer_name>]...

Field related options:
                [-addfields] [-relaxedFieldNameMatch]
                [-fieldTypeToString All|<type1>[,<type2>]...]
                [-mapFieldType <srctype>|All=<dsttype>[,<srctype2>=<dsttype2>]...]
                [-fieldmap <field_1>[,<field_2>]...] [-splitlistfields]
                [-maxsubfields <n>] [-emptyStrAsNull] [-forceNullable]
                [-unsetFieldWidth] [-unsetDefault] [-resolveDomains]
                [-dateTimeTo UTC|UTC(+|-)<HH>|UTC(+|-)<HH>:<MM>] [-noNativeData]

Advanced geometry and SRS related options:
                [-dim layer_dim|2|XY|3|XYZ|XYM|XYZM] [-s_coord_epoch <epoch>]
                [-a_coord_epoch <epoch>] [-t_coord_epoch <epoch>]
                [-ct <pipeline_def>] [-ct_opt <NAME>=<VALUE>]...
                [-spat_srs <srs_def>] [-geomfield <name>]
                [-segmentize <max_dist>] [-simplify <tolerance>] [-makevalid]
                [-skipinvalid] [-wrapdateline] [-datelineoffset <val_in_degree>]
                [-clipsrc [<xmin> <ymin> <xmax> <ymax>]|<WKT>|<datasource>|spat_extent]
                [-clipsrcsql <sql_statement>] [-clipsrclayer <layername>]
                [-clipsrcwhere <expression>]
                [-clipdst [<xmin> <ymin> <xmax> <ymax>]|<WKT>|<datasource>]
                [-clipdstsql <sql_statement>] [-clipdstlayer <layername>]
                [-clipdstwhere <expression>] [-explodecollections]
                [-zfield <name>]
                [-gcp <ungeoref_x> <ungeoref_y> <georef_x> <georef_y> [<elevation>]]...
                [-tps] [-order 1|2|3] [-xyRes <val>[ m|mm|deg]]
                [-zRes <val>[ m|mm]] [-mRes <val>] [-unsetCoordPrecision]

Other options:
                [--quiet] [-progress] [-if <format>]... [-oo <NAME>=<VALUE>]...
                [-doo <NAME>=<VALUE>]...
                [-fid <FID>] [-preserve_fid] [-unsetFid]
                [[-skipfailures]|[-gt <n>|unlimited]]
                [-limit <nb_features>] [-ds_transaction] [-mo <NAME>=<VALUE>]...
                [-nomd]
```

Description

ogr2ogr can be used to convert simple features data between file formats. It can also perform various operations during the process, such as spatial or attribute selection, reducing the set of attributes, setting the output coordinate system or even reprojecting the features during translation.

Tip

Equivalent in new "gdal" command line interface:

- *gdal vector clip* for spatial subsetting
- *gdal vector convert* for format translation

- *gdal vector explode-collections* to explode geometries of type collection
- *gdal vector make-valid* to fix geometry validity
- *gdal vector reproject* for reprojection
- *gdal vector segmentize* to segmentize geometries
- *gdal vector select* to select a subset of fields
- *gdal vector set-field-type* to change the type of an attribute field
- *gdal vector set-geom-type* to change the type of a geometry field
- *gdal vector simplify* to simplify geometries
- *gdal vector sql* to run SQL query

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-q, --quiet

Suppress progress messages and other non-error output.

-if <format>

Format/driver name to be attempted to open the input file(s). It is generally not necessary to specify it, but it can be used to skip automatic driver detection, when it fails to select the appropriate driver. This option can be repeated several times to specify several candidate drivers. Note that it does not force those drivers to open the dataset. In particular, some drivers have requirements on file extensions.

Added in version 3.2.

-of <format_name>, **-f** <format_name>

Output file format name, e.g. `ESRI Shapefile`, `MapInfo File`, `PostgreSQL`. If not specified, the format is guessed from the extension.

-append

Append to existing layer instead of creating new. This option also enables `-update`.

-upsert

Added in version 3.6.

Variant of `-append` where the *`OGRLayer::UpsertFeature()`* operation is used to insert or update features instead of appending with *`OGRLayer::CreateFeature()`*.

This is currently implemented only in a few drivers: *GPKG -- GeoPackage vector*, *Elasticsearch: Geographically Encoded Objects for Elasticsearch* and *MongoDBv3* (drivers that implement upsert expose the *`GDAL_DCAP_UPSERT`* capability).

The upsert operation uses the FID of the input feature, when it is set and is a "significant" (that is the FID column name is not the empty string), as the key to update existing features. It is crucial to make sure that the FID in the source and target layers are consistent.

For the GPKG driver, it is also possible to upsert features whose FID is unset or non-significant (`-unsetFid` can be used to ignore the FID from the source feature), when there is a UNIQUE column that is not the integer primary key.

-overwrite

Delete the output layer and recreate it empty

-update

Open existing output datasource in update mode rather than trying to create a new one

-select <field_list>

Comma-delimited list of fields from input layer to copy to the new layer.

Starting with GDAL 3.9, field names with spaces, commas or double-quote should be surrounded with a starting and ending double-quote character, and double-quote characters in a field name should be escaped with backslash.

Depending on the shell used, this might require further quoting. For example, to select `regular_field`, `a_field_with space, and comma` and `a field with " double quote` with a Unix shell:

```
-select "regular_field,\"a_field_with space, and comma\",\"a field with \\\" double␣
↪quote\""
```

A field is only selected once, even if mentioned several times in the list and if the input layer has duplicate field names.

Geometry fields can also be specified in the list.

All fields are selected when -select is not specified. Specifying the empty string can be used to disable selecting any attribute field, and only keep geometries.

Note this setting cannot be used together with `-append`. To control the selection of fields when appending to a layer, use `-fieldmap` or `-sql`.

-progress

Display progress on terminal. Only works if input layers have the "fast feature count" capability.

-sql <sql_statement>|@<filename>

SQL statement to execute. The resulting table/layer will be saved to the output. The `@filename` syntax can be used to indicate that the content is in the pointed filename. (Cannot be used with *-spat_srs*.)

-dialect <dialect>

SQL dialect. In some cases can be used to use the (unoptimized) *OGR SQL dialect* instead of the native SQL of an RDBMS by passing the `OGRSQL` dialect value. The *SQL SQLite dialect* dialect can be chosen with the `SQLITE` and `INDIRECT_SQLITE` dialect values, and this can be used with any datasource. (*-dialect* is ignored with *-where*. Use *-sql* instead.)

-where <restricted_where>|@<filename>

Attribute query (like SQL WHERE). The `@filename` syntax can be used to indicate that the content is in the pointed filename. (*-dialect* is ignored with *-where*. Use *-sql* instead.)

-skipfailures

Continue after a failure, skipping the failed feature. Note that this option overrides the value of *-gt* to 1, which may cause severe slowdown when inserting into databases.

-spat <xmin> <ymin> <xmax> <ymax>

spatial query extents, in the SRS of the source layer(s) (or the one specified with `-spat_srs`). Only features whose geometry intersects the extents will be selected. The geometries will not be clipped unless `-clipsrc` is specified.

-spat_srs <srs_def>

Override spatial filter SRS. (Cannot be used with *-sql*.)

-geomfield <field>

Name of the geometry field on which the spatial filter operates on.

-dsco <NAME>=<VALUE>

Dataset creation option (format specific)

-lco <NAME>=<VALUE>

Layer creation option (format specific)

-nln <name>

Assign an alternate name to the new layer

-nlt <type>

Define the geometry type for the created layer. One of NONE, GEOMETRY, POINT, LINESTRING, POLYGON, GEOMETRYCOLLECTION, MULTIPOINT, MULTIPOLYGON, MULTILINESTRING, CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE, and MULTISURFACE non-linear geometry types. Add Z, M, or ZM to the type name to specify coordinates with elevation, measure, or elevation and measure. PROMOTE_TO_MULTI can be used to automatically promote layers that mix polygon or multipolygons to multipolygons, and layers that mix linestrings or multilinestrings to multilinestrings. Can be useful when converting shapefiles to PostGIS and other target drivers that implement strict checks for geometry types. CONVERT_TO_LINEAR can be used to to convert non-linear geometry types into linear geometry types by approximating them, and CONVERT_TO_CURVE to promote a non-linear type to its generalized curve type (POLYGON to CURVEPOLYGON, MULTIPOLYGON to MULTISURFACE, LINESTRING to COMPOUNDCURVE, MULTILINESTRING to MULTICURVE). Starting with version 2.1 the type can be defined as measured ("25D" remains as an alias for single "Z"). Some forced geometry conversions may result in invalid geometries, for example when forcing conversion of multi-part multipolygons with -nlt POLYGON, the resulting polygon will break the Simple Features rules.

Starting with GDAL 3.0.5, -nlt CONVERT_TO_LINEAR and -nlt PROMOTE_TO_MULTI can be used simultaneously.

-dim <val>

Force the coordinate dimension to val (valid values are XY, XYZ, XYM, and XYZM - for backwards compatibility 2 is an alias for XY and 3 is an alias for XYZ). This affects both the layer geometry type, and feature geometries. The value can be set to layer_dim to instruct feature geometries to be promoted to the coordinate dimension declared by the layer.

-a_srs <srs_def>

Assign an output SRS, but without reprojecting (use *-t_srs* to reproject)

The coordinate reference systems that can be passed are anything supported by the *OGRSpatialReference::SetFromUserInput()* call, which includes EPSG Projected, Geographic or Compound CRS (i.e. EPSG:4296), a well known text (WKT) CRS definition, PROJ.4 declarations, or the name of a .prj file containing a WKT CRS definition.

-a_coord_epoch <epoch>

Added in version 3.4.

Assign a coordinate epoch, linked with the output SRS. Useful when the output SRS is a dynamic CRS. Only taken into account if *-a_srs* is used.

-t_srs <srs_def>

Reproject/transform to this SRS on output, and assign it as output SRS.

A source SRS must be available for reprojection to occur. The source SRS will be by default the one found in the source layer when it is available, or as overridden by the user with *-s_srs*

The coordinate reference systems that can be passed are anything supported by the *OGRSpatialReference::SetFromUserInput()* call, which includes EPSG Projected, Geographic or

Compound CRS (i.e. EPSG:4296), a well known text (WKT) CRS definition, PROJ.4 declarations, or the name of a .prj file containing a WKT CRS definition.

-t_coord_epoch <epoch>

Added in version 3.4.

Assign a coordinate epoch, linked with the output SRS. Useful when the output SRS is a dynamic CRS. Only taken into account if *-t_srs* is used. It is also mutually exclusive with *-a_coord_epoch*.

Before PROJ 9.4, *-s_coord_epoch* and *-t_coord_epoch* were mutually exclusive, due to lack of support for transformations between two dynamic CRS.

-s_srs <srs_def>

Override source SRS. If not specified the SRS found in the input layer will be used. This option has only an effect if used together with *-t_srs* to reproject.

The coordinate reference systems that can be passed are anything supported by the *OGRSpatialReference::SetFromUserInput()* call, which includes EPSG Projected, Geographic or Compound CRS (i.e. EPSG:4296), a well known text (WKT) CRS definition, PROJ.4 declarations, or the name of a .prj file containing a WKT CRS definition.

-xyRes "<val>[m|mm|deg]"

Added in version 3.9.

Set/override the geometry X/Y coordinate resolution. If only a numeric value is specified, it is assumed to be expressed in the units of the target SRS. The m, mm or deg suffixes can be specified to indicate that the value must be interpreted as being in meter, millimeter or degree.

When specifying this option, the *OGRGeometry::SetPrecision()* method is run on geometries (that are not curves) before passing them to the output driver, to avoid generating invalid geometries due to the potentially reduced precision (unless the *OGR_APPLY_GEOM_SET_PRECISION* configuration option is set to NO)

If neither this option nor *-unsetCoordPrecision* are specified, the coordinate resolution of the source layer, if available, is used.

-zRes "<val>[m|mm]"

Added in version 3.9.

Set/override the geometry Z coordinate resolution. If only a numeric value is specified, it is assumed to be expressed in the units of the target SRS. The m or mm suffixes can be specified to indicate that the value must be interpreted as being in meter or millimeter. If neither this option nor *-unsetCoordPrecision* are specified, the coordinate resolution of the source layer, if available, is used.

-mRes <val>

Added in version 3.9.

Set/override the geometry M coordinate resolution. If neither this option nor *-unsetCoordPrecision* are specified, the coordinate resolution of the source layer, if available, is used.

-unsetCoordPrecision

Added in version 3.9.

Prevent the geometry coordinate resolution from being set on target layer(s).

-s_coord_epoch <epoch>

Added in version 3.4.

Assign a coordinate epoch, linked with the source SRS. Useful when the source SRS is a dynamic CRS. Only taken into account if *-s_srs* is used.

Before PROJ 9.4, *-s_coord_epoch* and *-t_coord_epoch* were mutually exclusive, due to lack of support for transformations between two dynamic CRS.

-ct <string>

Added in version 3.0.

A PROJ string (single step operation or multiple step string starting with +proj=pipeline), a WKT2 string describing a CoordinateOperation, or a urn:ogc:def:coordinateOperation:EPSG::XXXX URN overriding the default transformation from the source to the target CRS.

It must take into account the axis order of the source and target CRS, that is typically include a `step proj=axisswap order=2,1` at the beginning of the pipeline if the source CRS has northing/easting axis order, and/or at the end of the pipeline if the target CRS has northing/easting axis order.

-ct_opt <NAME>=<VALUE>

Added in version 3.11.

Specify a coordinate operation option that influences how PROJ selects coordinate operations when `-ct` is *not* set.

The following options are available:

- `ONLY_BEST``=``YES/NO`. By default (at least in the PROJ 9.x series), PROJ may use coordinate operations that are not the "best" if resources (typically grids) needed to use them are missing. It will then fallback to other coordinate operations that have a lesser accuracy, for example using Helmert transformations, or in the absence of such operations, to ones with potential very rough accuracy, using "ballpark" transformations (see https://proj.org/glossary.html). When calling this method with YES, PROJ will only consider the "best" operation, and error out (at Transform() time) if they cannot be used. This method may be used together with `ALLOW_BALLPARK``=``NO` to only allow best operations that have a known accuracy. Note that this method has no effect on PROJ versions before 9.2. The default value for this option can be also set with the `PROJ_ONLY_BEST_DEFAULT` environment variable, or with the `only_best_default` setting of proj.ini. Setting ONLY_BEST=YES/NO overrides such default value.
- `ALLOW_BALLPARK``=``YES/NO`. Whether ballpark coordinate operations are allowed. Default is YES.
- `WARN_ABOUT_DIFFERENT_COORD_OP``=``YES/NO`. Can be set to NO to avoid GDAL warning when different coordinate operations are used to transform the different geometries of the dataset (or part of the same geometry). Default is YES.

-preserve_fid

Use the FID of the source features instead of letting the output driver automatically assign a new one (for formats that require a FID). If not in append mode, this behavior is the default if the output driver has a FID layer creation option, in which case the name of the source FID column will be used and source feature IDs will be attempted to be preserved. This behavior can be disabled by setting `-unsetFid`. This option is not compatible with `-explodecollections`.

-fid <fid>

If provided, only the feature with the specified feature id will be processed. Operates exclusive of the spatial or attribute queries. Note: if you want to select several features based on their feature id, you can also use the fact the 'fid' is a special field recognized by OGR SQL. So, *-where "fid in (1,3,5)"* would select features 1, 3 and 5.

-limit <nb_features>

Limit the number of features per layer.

-oo <NAME>=<VALUE>

Dataset open option (format specific).

If a driver supports the `OGR_SCHEMA` open option, it can be used to partially or completely override the auto-detected schema (i.e. which fields are read, with which types, subtypes, length, precision etc.) of the dataset.

The value of this option is a JSON string or a path to a JSON file that complies with the OGR_SCHEMA open option schema definition

-doo <NAME>=<VALUE>

Destination dataset open option (format specific), only valid in -update mode.

-gt <n>

Group n features per transaction (default 100 000). Increase the value for better performance when writing into DBMS drivers that have transaction support. n can be set to unlimited to load the data into a single transaction.

-ds_transaction

Force the use of a dataset level transaction (for drivers that support such mechanism), especially for drivers such as FileGDB that only support dataset level transaction in emulation mode.

-clipsrc [<xmin> <ymin> <xmax> <ymax>]|WKT|<datasource>|spat_extent

Clip geometries (before potential reprojection) to one of the following:

- the specified bounding box (expressed in source SRS)
- a WKT geometry (POLYGON or MULTIPOLYGON expressed in source SRS)
- one or more geometries selected from a datasource
- the spatial extent of the -spat option if you use the spat_extent keyword.

When specifying a datasource, you will generally want to use -clipsrc in combination of the -clipsrclayer, -clipsrcwhere or -clipsrcsql options.

-clipsrcsql <sql_statement>

Select desired geometries from the source clip datasource using an SQL query.

-clipsrclayer <layername>

Select the named layer from the source clip datasource.

-clipsrcwhere <expression>

Restrict desired geometries from the source clip layer based on an attribute query.

-clipdst [<xmin> <ymin> <xmax> <ymax>]|<WKT>|<datasource>

Clip geometries (after potential reprojection) to one of the following:

- the specified bounding box (expressed in destination SRS)
- a WKT geometry (POLYGON or MULTIPOLYGON expressed in destination SRS)
- one or more geometries selected from a datasource

When specifying a datasource, you will generally want to use -clipdst in combination with the -clipdstlayer, -clipdstwhere or -clipdstsql options.

-clipdstsql <sql_statement>

Select desired geometries from the destination clip datasource using an SQL query.

-clipdstlayer <layername>

Select the named layer from the destination clip datasource.

-clipdstwhere <expression>

Restrict desired geometries from the destination clip layer based on an attribute query.

-wrapdateline

Split geometries crossing the dateline meridian (long. = +/- 180deg)

-datelineoffset

Offset from dateline in degrees (default long. = +/- 10deg, geometries within 170deg to -170deg will be split)

-simplify <tolerance>

Distance tolerance for simplification. Note: the algorithm used preserves topology per feature, in particular for polygon geometries, but not for a whole layer.

The specified value of this option is the tolerance used to merge consecutive points of the output geometry using the *OGRGeometry::SimplifyPreserveTopology()* method The unit of the distance is in georeferenced units of the source vector dataset. This option is applied before the reprojection implied by *-t_srs*

-segmentize <max_dist>

The specified value of this option is the maximum distance between two consecutive points of the output geometry before intermediate points are added. The unit of the distance is georeferenced units of the source layer. This option is applied before the reprojection implied by *-t_srs*

-makevalid

Run the *OGRGeometry::MakeValid()* operation, followed by *OGRGeometryFactory::removeLowerDimensionSubGeoms()*, on geometries to ensure they are valid regarding the rules of the Simple Features specification.

Added in version 3.1: (requires GEOS)

-skipinvalid

Run the *OGRGeometry::IsValid()* operation on geometries to check if they are valid regarding the rules of the Simple Features specification. If they are not, the feature is skipped. This check is done after all other geometry operations.

Added in version 3.10: (requires GEOS)

-fieldTypeToString All|<type1>[,<type2>]...

Converts any field of the specified type to a field of type string in the destination layer. Valid types are : Integer, Integer64, Real, String, Date, Time, DateTime, Binary, IntegerList, Integer64List, RealList, StringList. Special value All can be used to convert all fields to strings. This is an alternate way to using the CAST operator of OGR SQL, that may avoid typing a long SQL query. Note that this does not influence the field types used by the source driver, and is only an afterwards conversion. Also note that this option is without effects on fields whose presence and type is hard-coded in the output driver (e.g KML, GPX). For an alternative way to manipulate field types earlier in the process while they are read from the input dataset see *-oo* OGR_SCHEMA (only available for a limited set of formats).

-mapFieldType {<srctype>|All=<dsttype>[,<srctype2>=<dsttype2>]...}

Converts any field of the specified type to another type. Valid types are : Integer, Integer64, Real, String, Date, Time, DateTime, Binary, IntegerList, Integer64List, RealList, StringList. Types can also include subtype between parenthesis, such as Integer(Boolean), Real(Float32), ... Special value All can be used to convert all fields to another type. This is an alternate way to using the CAST operator of OGR SQL, that may avoid typing a long SQL query. This is a generalization of -fieldTypeToString. Note that this does not influence the field types used by the source driver, and is only an afterwards conversion. Also note that this option is without effects on fields whose presence and type is hard-coded in the output driver (e.g KML, GPX). For an alternative way to manipulate field types earlier in the process while they are read from the input dataset see *-oo* OGR_SCHEMA (only available for a limited set of formats).

-dateTimeTo {UTC|UTC(+|-)<HH>|UTC(+|-)<HH>:<MM>}

Added in version 3.7.

Converts date time values from the timezone specified in the source value to the target timezone expressed with *-dateTimeTo*. Datetime whose timezone is unknown or localtime are not modified.

HH must be in the [0,14] range and MM=00, 15, 30 or 45.

-unsetFieldWidth

Set field width and precision to 0.

-splitlistfields

Split fields of type StringList, RealList or IntegerList into as many fields of type String, Real or Integer as necessary.

-maxsubfields <val>

To be combined with `-splitlistfields` to limit the number of subfields created for each split field.

-explodecollections

Produce one feature for each geometry in any kind of geometry collection in the source file, applied after any `-sql` option. This options is not compatible with `-preserve_fid` but `-sql "SELECT fid AS original_fid, * FROM ..."` can be used to store the original FID if needed.

-zfield <field_name>

Uses the specified field to fill the Z coordinate of geometries.

-gcp <ungeoref_x> <ungeoref_y> <georef_x> <georef_y> [<elevation>]

Use the indicated ground control point to compute a coordinate transformation. The transformation method can be selected by specifying the *-order* or *-tps* options. Note that unlike raster tools such as gdal_edit or gdal_translate, GCPs are not added to the output dataset. This option may be provided multiple times to provide a set of GCPs (at least 2 GCPs are needed).

-order <n>

Order of polynomial used for warping (1 to 3). The default is to select a polynomial order based on the number of GCPs.

-tps

Force use of thin plate spline transformer based on available GCPs.

-fieldmap

Specifies the list of field indexes to be copied from the source to the destination. The (n)th value specified in the list is the index of the field in the target layer definition in which the n(th) field of the source layer must be copied. Index count starts at zero. To omit a field, specify a value of -1. There must be exactly as many values in the list as the count of the fields in the source layer. We can use the 'identity' setting to specify that the fields should be transferred by using the same order. This setting should be used along with the `-append` setting.

-addfields

This is a specialized version of `-append`. Contrary to `-append`, `-addfields` has the effect of adding, to existing target layers, the new fields found in source layers. This option is useful when merging files that have non-strictly identical structures. This might not work for output formats that don't support adding fields to existing non-empty layers. Note that if you plan to use -addfields, you may need to combine it with -forceNullable, including for the initial import.

-relaxedFieldNameMatch

Do field name matching between source and existing target layer in a more relaxed way if the target driver has an implementation for it.

-forceNullable

Do not propagate not-nullable constraints to target layer if they exist in source layer.

-unsetDefault

Do not propagate default field values to target layer if they exist in source layer.

-unsetFid

Can be specified to prevent the name of the source FID column and source feature IDs from being reused for the target layer. This option can for example be useful if selecting source features with a ORDER BY clause.

-emptyStrAsNull

Added in version 3.3.

Treat empty string values as null.

-resolveDomains

Added in version 3.3.

When this is specified, any selected field that is linked to a coded field domain will be accompanied by an additional field (`{dstfield}_resolved`), that will contain the description of the coded value.

-nomd

To disable copying of metadata from source dataset and layers into target dataset and layers, when supported by output driver.

-mo `<META-TAG>=<VALUE>`

Passes a metadata key and value to set on the output dataset, when supported by output driver.

-noNativeData

To disable copying of native data, i.e. details of source format not captured by OGR abstraction, that are otherwise preserved by some drivers (like GeoJSON) when converting to same format.

<dst_dataset_name>

Output dataset name.

<src_dataset_name>

Source dataset name.

<layer_name>

One or more source layer names to copy to the output dataset. If no layer names are passed, then all source layers are copied.

Performance Hints

When writing into transactional DBMS (SQLite/PostgreSQL,MySQL, etc...), it might be beneficial to increase the number of INSERT statements executed between BEGIN TRANSACTION and COMMIT TRANSACTION statements. This number is specified with the -gt option. For example, for SQLite, explicitly defining -gt 65536 ensures optimal performance while populating some table containing many hundreds of thousands or millions of rows. However, note that -skipfailures overrides -gt and sets the size of transactions to 1.

For PostgreSQL, the *PG_USE_COPY* config option can be set to YES for a significant insertion performance boost. See the PG driver documentation page.

More generally, consult the documentation page of the input and output drivers for performance hints.

Known issues

Starting with GDAL 3.8, ogr2ogr uses internally an Arrow array based API (cf rfc-86) for some source formats (in particular GeoPackage or FlatGeoBuf), and for the most basic types of operations, to improve performance. This substantial change in the ogr2ogr internal logic has required a number of fixes throughout the GDAL 3.8.x bugfix releases to fully stabilize it, and we believe most issues are resolved with GDAL 3.9. If you hit errors not met with earlier GDAL versions, you may specify `--config OGR2OGR_USE_ARROW_API NO` on the ogr2ogr command line to opt for the classic algorithm using an iterative feature based approach. If that flag is needed with GDAL >= 3.9, please file an issue on the GDAL issue tracker.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

C API

This utility is also callable from C with *GDALVectorTranslate()*.

Examples

Example 1: Appending to an existing layer

```
ogr2ogr -append -f PostgreSQL PG:dbname=warmerda abc.tab
```

Example 2: Clip input layer with a bounding box

The bounding box is specified as `<xmin> <ymin> <xmax> <ymax>`:

```
ogr2ogr -spat -13.931 34.886 46.23 74.12 output.gpkg natural_earth_vector.gpkg
```

Example 3: Filter Features by a -where clause

```
ogr2ogr -where "\"POP_EST\" < 1000000" \
  output.gpkg natural_earth_vector.gpkg ne_10m_admin_0_countries
```

More examples are given in the individual format pages.

Advanced examples

Example 4: Reprojecting from ETRS_1989_LAEA_52N_10E to EPSG:4326 and clipping to a bounding box

```
ogr2ogr -wrapdateline -t_srs EPSG:4326 -clipdst -5 40 15 55 france_4326.shp␣
↪europe_laea.shp
```

Example 5: Outputting geometries with the CSV driver.

By default this driver does not preserve geometries upon layer creation. An explicit layer creation option is needed:

```
ogr2ogr -lco GEOMETRY=AS_XYZ TrackWaypoint.csv TrackWaypoint.kml
```

Example 6: Extracting only geometries

There are different situations, depending if the input layer has a named geometry column, or not. First check, with **ogrinfo** if there is a reported geometry column.

```
$ ogrinfo -so CadNSDI.gdb.zip PLSSPoint | grep 'Geometry Column'
Geometry Column = SHAPE
```

In that situation where the input format is a FileGeodatabase, it is called SHAPE and can thus be referenced directly in a SELECT statement.

```
ogr2ogr -sql "SELECT SHAPE FROM PLSSPoint" \
  -lco GEOMETRY=AS_XY -f CSV /vsistdout/ CadNSDI.gdb.zip
```

For a shapefile with a unnamed geometry column, `_ogr_geometry_` can be used as a special name to designate the implicit geometry column, when using the default *OGR SQL* dialect. The name begins with an underscore and SQL syntax requires that it must appear between double quotes. In addition the command line interpreter may require that double quotes are escaped and the final SELECT statement could look like:

```
ogr2ogr -sql "SELECT \"_ogr_geometry_\" FROM PLSSPoint" \
  -lco GEOMETRY=AS_XY -f CSV /vsistdout/ CadNSDI.shp
```

If using the *SQL SQLite* dialect, the special geometry name is `geometry` when the source geometry column has no name.

```
ogr2ogr -sql "SELECT geometry FROM PLSSPoint" -dialect SQLite \
  -lco GEOMETRY=AS_XY -f CSV /vsistdout/ CadNSDI.shp
```

5.2.4.4 ogrtindex

Synopsis

```
Usage: ogrtindex [--help] [--long-usage] [--help-general]
                 [-lnum <n>]... [-lname <name>]... [-of <output_format>]
                 [-tileindex <tileindex>] [-write_absolute_path]
                 [-skip_different_projection] [-t_srs <srs_def>]
                 [-src_srs_name <field_name>]
                 [-src_srs_format {AUTO|WKT|EPSG|PROJ}]
                 [-accept_different_schemas]
                 <output_dataset> <src_dataset>...
```

Description

ogrtindex program can be used to create a tileindex - a file containing a list of the identities of a bunch of other files along with their spatial extents. This is primarily intended to be used with MapServer for tiled access to layers using the OGR connection type.

> **Tip**
>
> Equivalent in new "gdal" command line interface:
>
> See *gdal vector index*.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-lnum <n>

Add layer number `n` from each source file in the tile index.

-lname <name>

Add the layer named `name` from each source file in the tile index.

-f <output_format>

Select an output format name. The default is to create a shapefile.

-tileindex <field_name>

The name to use for the dataset name. Defaults to LOCATION.

-write_absolute_path

Filenames are written with absolute paths

-skip_different_projection

Only layers with same projection ref as layers already inserted in the tileindex will be inserted.

-t_srs <target_srs>

Extent of input files will be transformed to the desired target coordinate reference system. Using this option generates files that are not compatible with MapServer < 7.2. Default creates simple rectangular polygons in the same coordinate reference system as the input vector layers.

-src_srs_name <field_name>

The name of the field to store the SRS of each tile. This field name can be used as the value of the TILESRS keyword in MapServer >= 7.2.

-src_srs_format {AUTO|WKT|EPSG|PROJ}

The format in which the SRS of each tile must be written. Available formats are: AUTO, WKT, EPSG, PROJ.

-accept_different_schemas

By default ogrtindex checks that all layers inserted into the index have the same attribute schemas. If you specify this option, this test will be disabled. Be aware that resulting index may be incompatible with MapServer!

If no *-lnum* or *-lname* arguments are given it is assumed that all layers in source datasets should be added to the tile index as independent records.

If the tile index already exists it will be appended to, otherwise it will be created.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1

This example would create a shapefile (`tindex.shp`) containing a tile index of the `BL2000_LINK` layers in all the NTF files in the `wrk` directory:

```
ogrtindex tindex.shp wrk/*.NTF
```

5.2.4.5 ogrlineref

Synopsis

```
ogrlineref [--help] [--help-general] [-progress] [-quiet]
           [-f <format_name>] [-dsco <NAME>=<VALUE>]... [-lco <NAME>=<VALUE>]...
           [-create]
           [-l <src_line_datasource_name>] [-ln <layer_name>] [-lf <field_name>]
           [-p <src_repers_datasource_name>] [-pn <layer_name>] [-pm <pos_field_name>] [-
```

(continues on next page)

(continued from previous page)

```
↪pf <field_name>]
          [-r <src_parts_datasource_name>] [-rn <layer_name>]
          [-o <dst_datasource_name>] [-on <layer_name>] [-of <field_name>] [-s <step>]
          [-get_pos] [-x <long>] [-y <lat>]
          [-get_coord] [-m <position>]
          [-get_subline] [-mb <position>] [-me <position>]
```

Description

The **ogrlineref** program can be used to:

- create a linear reference file from input data
- return the "linear referenced" distance for the projection of the input coordinates (point) on the path
- return the coordinates (point) on the path according to the "linear referenced" distance
- return the portion of the path according to the "linear referenced" begin and end distances

The **ogrlineref** utility creates a linear reference - a file containing segments of a certain length (e.g. 1 km in reference units.) The user can get coordinates, linear referenced distances or sublines (subpaths) from this file. The utility does not require the `M` or `Z` components in the geometry. The results can be stored in any OGR supported format. Also some information is written to the stdout.

--help
: Show this help message and exit

--help-general
: Gives a brief usage message for the generic GDAL commandline options and exit.

-progress
: Show progress.

-q, --quiet
: Suppress progress messages and other non-error output.

-f `<format_name>`
: Select an output format name. The default is to create a shapefile.

-dsco `<NAME>=<VALUE>`
: Dataset creation option (format specific)

-lco `<NAME>=<VALUE>`
: Layer creation option (format specific).

-create
: Create the linear reference file (linestring of parts).

-l `<src_line_datasource_name>`
: The path to input linestring datasource (e.g. the road)

-ln `<layer_name>`
: The layer name in datasource

-lf `<field_name>`
: The field name of unique values to separate the input lines (e.g. the set of roads).

-p <src_repers_datasource_name>
The path to linear references points (e.g. the road mile-stones)

-pn <layer_name>
The layer name in datasource

-pm <pos_field_name>
The field name of distances along path (e.g. mile-stones values)

-pf <field_name>
The field name of unique values to map input reference points to lines.

-r <src_parts_datasource_name>
The path to linear reference file.

-rn <layer_name>
The layer name in datasource

-o <dst_datasource_name>
The path to output linear reference file (linestring datasource)

-on <layer_name>
The layer name in datasource

-of <field_name>
The field name for storing the unique values of input lines

-s <step>
The part size in linear units

-get_pos
Return linear referenced position for input X, Y

-x <long>
Input X coordinate

-y <lat>
Input Y coordinate

-get_coord
Return point on path for input linear distance

-m <position>
The input linear distance

-get_subline
Return the portion of the input path from and to input linear positions.

-mb <position>
The input begin linear distance

-me <position>
The input end linear distance

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1

This example would create a shapefile (`parts.shp`) containing a data needed for linear referencing (1 km parts):

```
ogrlineref -create -l roads.shp -p references.shp -pm dist -o parts.shp -s 1000 -progress
```

5.2.4.6 ogrmerge

Synopsis

```
ogrmerge [--help] [--help-general]
            -o <out_dsname> <src_dsname> [<src_dsname>]...
            [-f format] [-single] [-nln <layer_name_template>]
            [-update | -overwrite_ds] [-append | -overwrite_layer]
            [-src_geom_type <geom_type_name>[,<geom_type_name>]...]
            [-dsco <NAME>=<VALUE>]... [-lco <NAME>=<VALUE>]...
            [-s_srs <srs_def>] [-t_srs <srs_def> | -a_srs <srs_def>]
            [-progress] [-skipfailures] [--help-general]
```

Options specific to the *-single* option:

```
[-field_strategy FirstLayer|Union|Intersection]
[-src_layer_field_name <name>]
[-src_layer_field_content <layer_name_template>]
```

Description

ogrmerge takes as input several vector datasets, each of them having one or several vector layers, and copies them into a target dataset.

There are essentially two modes:

- the default one, where each input vector layer, is copied as a separate layer into the target dataset
- another one, activated with the -single switch, where the content of all input vector layers is appended into a single target layer. This assumes that the schema of those vector layers is more or less the same.

Internally this generates a *VRT -- OGR Virtual Format* file, and if the output format is not VRT, final translation is done with **ogr2ogr** or `gdal.VectorTranslate()`. So, for advanced uses, output to VRT, potential manual editing of it and **ogr2ogr** can be done.

> **Note**
>
> ogrmerge is a Python utility, and is only available if GDAL Python bindings are available.

> **Tip**
>
> Equivalent in new "gdal" command line interface:
>
> See *gdal vector concat*.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-o <out_dsname>

Output dataset name. Required.

<src_dsname>

One or several input vector datasets. Required

-f <format>

Select the output format. If not specified, the format is guessed from the extension (previously was ESRI Shapefile). Use the short format name

-single

If specified, all input vector layers will be merged into a single one.

-nln <layer_name_template>

Name of the output vector layer (in single mode, and the default is "merged"), or template to name the output vector layers in default mode (the default value is {AUTO_NAME}). The template can be a string with the following variables that will be substituted with a value computed from the input layer being processed:

- {AUTO_NAME}: equivalent to {DS_BASENAME}_{LAYER_NAME} if both values are different, or {LAYER_NAME} when they are identical (case of shapefile).
- {DS_NAME}: name of the source dataset
- {DS_BASENAME}: base name of the source dataset
- {DS_INDEX}: index of the source dataset
- {LAYER_NAME}: name of the source layer
- {LAYER_INDEX}: index of the source layer

-update

Open an existing dataset in update mode.

-overwrite_ds

Overwrite the existing dataset if it already exists (for file based datasets)

-append

Open an existing dataset in update mode, and if output layers already exist, append the content of input layers to them.

-overwrite_layer

Open an existing dataset in update mode, and if output layers already exist, replace their content with the one of the input layer.

-src_geom_type <geom_type_name>[,<geom_type_name>]...

Only take into account input layers whose geometry type match the type(s) specified. Valid values for geom_type_name are GEOMETRY, POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRYCOLLECTION, CIRCULARSTRING, CURVEPOLYGON, MULTICURVE, MULTISURFACE, CURVE, SURFACE, TRIANGLE, POLYHEDRALSURFACE and TIN.

-dsco <NAME>=<VALUE>

Dataset creation option (format specific)

-lco <NAME>=<VALUE>

Layer creation option (format specific)

-a_srs <srs_def>

Assign an output SRS

-t_srs <srs_def>

Reproject/transform to this SRS on output

-s_srs <srs_def>

Override source SRS

-progress

Display progress on terminal. Only works if input layers have the "fast feature count" capability.

-skipfailures

Continue after a failure, skipping the failed feature.

-field_strategy `FirstLayer|Union|Intersection`

Only used with *-single*. Determines how the schema of the target layer is built from the schemas of the input layers. May be FirstLayer to use the fields from the first layer found, Union to use a super-set of all the fields from all source layers, or Intersection to use a sub-set of all the common fields from all source layers. Defaults to Union.

-src_layer_field_name <name>

Only used with *-single*. If specified, the schema of the target layer will be extended with a new field 'name', whose content is determined by -src_layer_field_content. See *Example 3*.

-src_layer_field_content <layer_name_template>

Only used with *-single*. If specified, the schema of the target layer will be extended with a new field (whose name is given by *-src_layer_field_name*, or 'source_ds_lyr' otherwise), whose content is determined by `layer_name_template`. The syntax of `layer_name_template` is the same as for *-nln*.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

Examples

Example 1: Creating a VRT with a layer for each input shapefile

```
ogrmerge -o merged.vrt *.shp
```

Example 2: Creating a GeoPackage with a layer for each input shapefile

```
ogrmerge -o merged.gpkg *.shp
```

Example 3: Adding a field to indicate the source layer

Concatenate the content of `france.shp` and `germany.shp` in `merged.shp`, and add a 'country' field to each feature whose value is 'france' or 'germany' depending where it comes from:

```
ogrmerge -single -o merged.shp france.shp germany.shp -src_layer_field_name country
```

5.2.4.7 ogr_layer_algebra

Added in version 3.6.

Synopsis

```
ogr_layer_algebra [--help] [--help-general]
                    Union|Intersection|SymDifference|Identity|Update|Clip|Erase
                    -input_ds <path> [-input_lyr <name>]
                    -method_ds <path> [-method_lyr <name>]
                    -output_ds <path> [-output_lyr <name>] [-overwrite]
                    [-opt <NAME>=<VALUE>]...
                    [-f <format_name>] [-dsco <NAME>=<VALUE>]... [-lco <NAME>=<VALUE>]...
                    [-input_fields {NONE|ALL|<fld1>,<fl2>,...<fldN>}] [-method_fields
↪{NONE|ALL|<fld1>,<fl2>,...<fldN>}]
                    [-nlt <geom_type>] [-a_srs <srs_def>]
```

Description

The **ogr_layer_algebra** provides a command line utility to perform various vector layer algebraic operations. The utility takes a vector input source and a method source and generates the output of the operation in the specified output file.

> **Note**
>
> ogr_layer_algebra is a Python utility, and is only available if GDAL Python bindings are available.

> **Tip**
>
> Equivalent in new "gdal" command line interface:
>
> See *gdal vector layer-algebra*.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

<mode>

Where <mode> is one of the seven available modes:

- `Union`

  A union is a set of features, which represent areas that are in either of the operand layers.

- `Intersection`

  An intersection is a set of features, which represent the common areas of two layers.

- `SymDifference`

A symmetric difference is a set of features, which represent areas that are in operand layers but which do not intersect.

- `Identity`

  The identity method identifies features in the input layer with features in the method layer possibly splitting features into several features. By default the result layer has attributes from both operand layers.

- `Update`

  The update method creates a layer, which add features into the input layer from the method layer possibly cutting features in the input layer. By default the result layer has attributes only from the input layer.

- `Clip`

  The clip method creates a layer, which has features from the input layer clipped to the areas of the features in the method layer. By default the result layer has attributes of the input layer.

- `Erase`

  The erase method creates a layer, which has features from the input layer whose areas are erased by the features in the method layer. By default the result layer has attributes of the input layer.

-input_ds <path>

Input dataset path for the operation to be performed. For operations involving two datasets, this is one of the datasets.

-input_lyr <name>

Layer name of the `input_ds` for which the operations have to be performed (Optional)

-method_ds <path>

Method data set path for the operation to be performed. This is usually the conditional data set supplied to the operation (ex: clip , erase , update) This is the Second data set in the operation (ex : Union, Intersection , SymDifference)

-method_lyr <name>

Layer name of the `method_ds` for which the operations have to be performed (Optional)

-output_ds <path>

Output data set path for writing the result of the operations performed by `ogr_layer_algebra`.

-output_lyr_name <name>

Layer name of the `output_lyr_name` where the output vector has to be written. (Optional)

-overwrite

Indicates whether the `output_ds` have to be overwritten with the generated result of `ogr_layer_algebra`.

-opt <NAME>=<VALUE>

Attributes for which the operation has to run on `input_ds` and `method_ds`.

-f <format_name>

Select the output format.If not specified, the format is guessed from the extension (previously was ESRI Shapefile). Use the short format name

-dsco <NAME>=<VALUE>

Dataset creation option (format specific).

-lco <NAME>=<VALUE>

Layer creation option (format specific).

-input_fields {NONE|ALL|<fld1>,<fl2>,...<fldN>}

Comma-delimited list of fields from input layer to copy to the output layer , if eligible according to the operation.

-method_fields {NONE|ALL|<fld1>,<fl2>,...<fldN>}

Comma-delimited list of fields from method layer to copy to the output layer , if eligible according to the operation.

-nlt <geom_type>

Define the geometry type for the created layer. One of NONE, GEOMETRY, POINT, LINESTRING, POLYGON, GEOMETRYCOLLECTION, MULTIPOINT, MULTIPOLYGON, GEOMETRY25D, POINT25D, LINESTRING25D, POLYGON25D, GEOMETRYCOLLECTION25D, MULTIPOINT25D, MULTIPOLYGON25D.

-a_srs <srs_def>

Assign an output SRS, but without reprojecting

The coordinate reference systems that can be passed are anything supported by the OGRSpatialReference.SetFromUserInput() call, which includes EPSG Projected, Geographic or Compound CRS (i.e. EPSG:4296), a well known text (WKT) CRS definition, PROJ.4 declarations, or the name of a .prj file containing a WKT CRS definition.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

5.2.5 Geographic network programs

5.2.5.1 gnmmanage

Synopsis

```
Usage: gnmmanage [--help][--help-general][-q][-quiet][--long-usage]
                 [info]
                 [create [-f <format_name>] [-t_srs <srs_name>] [-dsco <NAME>=<VALUE>]...
↪ ]
                 [import <src_dataset_name>] [-l <layer_name>]
                 [connect <gfid_src> <gfid_tgt> <gfid_con> [-c <cost>] [-ic <inv_cost>]␣
↪[-dir <dir>]]
                 [disconnect <gfid_src> <gfid_tgt> <gfid_con>]
                 [rule <rule_str>]
                 [autoconnect <tolerance>]
                 [delete]
                 [change [-bl <gfid>][-unbl <gfid>][-unblall]]
                 <gnm_name> [<layer> [<layer>]...]
```

Description

The **gnmmanage** program can perform various managing operations on geographical networks in GDAL. In addition to creating and deleting networks this includes capabilities of managing network's features, topology and rules.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

-info

Different information about network: system and class layers, network metadata, network spatial reference.

create

Create network.

-f <format_name>

Output file format name.

-t_srs <srs_name>

Spatial reference input.

-dsco <NAME>=<VALUE>

Network creation option set as pair name=value.

import <src_dataset_name>

Import layer with dataset name to copy.

-l <layer_name>

Layer name in dataset. If unset, 0 layer is copied.

connect <gfid_src> <gfid_tgt> <gfid_con>

Make a topological connection, where the gfid_src and gfid_tgt are vertices and gfid_con is edge (gfid_con can be -1, so the system edge will be inserted).

Manually assign the following values:

-c <cost>

Cost / weight

-ic <invcost>

Inverse cost

-dir <dir>

Direction of the edge.

disconnect <gfid_src> <gfid_tgt> <gfid_con>

Removes the connection from the graph.

rule <rule_str>

Creates a rule in the network by the given rule_str string.

autoconnect <tolerance>

Create topology automatically with the given double tolerance and layer names. In no layer name provided all layers of network will be used.

delete

Delete network.

change

Change blocking state of network edges or vertices.

-bl <gfid>

Block feature before the main operation. Blocking features are saved in the special layer.

-unbl <gfid>

Unblock feature before the main operation.

-unblall

Unblock all blocked features before the main operation.

<gnm_name>

The network to work with (path and name).

<layer>

The network layer name.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

5.2.5.2 gnmanalyse

Synopsis

```
Usage: gnmanalyse [--help][--help-general][-q][-quiet][--long-usage]
                  [dijkstra <start_gfid> <end_gfid [-alo <NAME>=<VALUE>]...]
                  [kpaths <start_gfid> <end_gfid> <k> [-alo NAME=VALUE]...]
                  [resource [-alo <NAME>=<VALUE>]...]
                  [-ds <ds_name>][-f <ds_format>][-l <layer_name>]
                  [-dsco <NAME>=<VALUE>]... [-lco <NAME>=<VALUE>]...
                  <gnm_name>
```

Description

The **gnmanalyse** program provides analysing capabilities of geographical networks in GDAL. The results of calculations are return in an OGRLayer format or as a console text output if such layer is undefined. All calculations are made considering the blocking state of features.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

dijkstra <start_gfid> <end_gfid>

Calculates the best path between two points using Dijkstra algorithm from start_gfid point to end_gfid point.

kpaths <start_gfid> <end_gfid>

Calculates K shortest paths between two points using Yen's algorithm (which internally uses Dijkstra algorithm for single path calculating) from start_gfid point to end_gfid point.

resource

Calculates the "resource distribution". The connected components search is performed using breadth-first search and starting from that features which are marked by rules as 'EMITTERS'.

-d <ds_name>

The name and path of the dataset to save the layer with resulting paths. Not need to be existed dataset.

-f <ds_format>

Define this to set the format of newly created dataset.

-l <layer_name>

The name of the resulting layer. If the layer exist already - it will be rewritten.

<gnm_name>

The network to work with (path and name).

-dsco <NAME>=<VALUE>

Dataset creation option (format specific)

-lco <NAME>=<VALUE>

Layer creation option (format specific)

-alo <NAME>=<VALUE>

Algorithm option (format specific)

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

5.2.6 Other utilities

5.2.6.1 sozip

Added in version 3.7.

Synopsis

```
sozip [--help] [--help-general]
      [--quiet|--verbose]
      [[-g|--grow] | [--overwrite]]
      [-r|--recurse-paths]
      [-j|--junk-paths]
      [-l|--list]
      [--optimize-from=<input.zip>]
      [--validate]
      [--enable-sozip={auto|yes|no}]
      [--sozip-chunk-size=<value>]
      [--sozip-min-file-size=<value>]
      [--content-type=<value>]
      <zip_filename> [<filename>]...
```

Description

The **sozip** utility can be used to:

- create a *SOZip (Seek-Optimized ZIP)* file
- append files to an existing ZIP/SOZip file
- list the contents of a ZIP/SOZip file

- validate a SOZip file
- convert an existing Zip file in a SOZip optimized one

Tip

Equivalent in new "gdal" command line interface:

See *gdal vsi sozip*.

--help

Show this help message and exit

--help-general

Gives a brief usage message for the generic GDAL commandline options and exit.

--quiet

Quiet mode. No progress message is emitted on the standard output.

--verbose

Verbose mode.

-g

--grow

Grow an existing zip file with the content of the specified filename(s). This is the default mode of the utility. This switch is here for compatibility with Info-ZIP **zip** utility

--overwrite

Overwrite the target zip file if it already exists.

-l

--list

List the files contained in the zip file in an output similar to Info-ZIP **unzip** utility, but with the addition of a column indicating whether each file is seek-optimized.

--validate

Validates a SOZip file. Baseline ZIP validation is done in a light way, limited to being able to browse through ZIP records with the InfoZIP-based ZIP reader used by GDAL. But validation of the SOZip-specific aspects is done more thoroughly.

-r

--recurse-paths

Travels the directory structure of the specified directory/directories recursively.

-j

--junk-paths

Store just the name of a saved file (junk the path), and do not store directory names. By default, sozip will store the full path (relative to the current directory).

--optimize-from=<input.zip>

Re-process {input.zip} to generate a SOZip-optimized .zip. Options *--enable-sozip*, *--sozip-chunk-size* and *--sozip-min-file-size* may be used in that mode.

--enable-sozip={auto|yes|no}

In auto mode, a file is seek-optimized only if its size is above the value of *--sozip-min-file-size*. In yes mode, all input files will be seek-optimized. In no mode, no input files will be seek-optimized.

--sozip-chunk-size=<value>

Chunk size for a seek-optimized file. Defaults to 32768 bytes. The value is specified in bytes, or K and M suffix can be respectively used to specify a value in kilo-bytes or mega-bytes.

--sozip-min-file-size=<value>

Minimum file size to decide if a file should be seek-optimized, in --enable-sozip=auto mode. Defaults to 1 MB byte. The value is specified in bytes, or K, M or G suffix can be respectively used to specify a value in kilo-bytes, mega-bytes or giga-bytes.

--content-type=<value>

Store the Content-Type for the file being added as a key-value pair in the extra field extension 'KV' (0x564b) dedicated to storing key-value pair metadata

<zip_filename>

Filename of the zip file to create/append to/list.

<filename>

Filename of the file to add.

Multithreading

The *GDAL_NUM_THREADS* configuration option can be set to ALL_CPUS or a integer value to specify the number of threads to use for SOZip-compressed files. Defaults to ALL_CPUS.

Return status code

The program returns status code 0 in case of success, and non-zero in case of error (non-blocking errors emitted as warnings are considered as a successful execution).

C API

Functionality of this utility can be done from C with *CPLAddFileInZip()* or *VSICopyFile()*.

Examples

Create a, potentially seek-optimized, ZIP file with the content of my.gpkg:

```
sozip my.gpkg.zip my.gpkg
```

Create a, potentially seek-optimized, ZIP file from the content of a source directory:

```
sozip -r my.gpkg.zip source_dir/
```

Create a, potentially seek-optimized, ZIP file from an existing ZIP file.

```
sozip --convert-from=in.zip out.zip
```

List the contents of a ZIP file and display which files are seek-optimized:

```
sozip -l my.gpkg.zip
```

Validates a SOZip file:

```
sozip --validate my.gpkg.zip
```

CHAPTER

SIX

RASTER DRIVERS

Short Name	Long Name	Creation (1)	Copy (2)	Georef
AAIGrid	Arc/Info ASCII Grid	No	**Yes**	**Yes**
ACE2	ACE2	No	No	**Yes**
ADRG	ADRG/ARC Digitized Raster Graphics (.gen/.thf)	No	No	**Yes**
AIG	Arc/Info Binary Grid	No	No	**Yes**
AIRSAR	AIRSAR Polarimetric Format	No	No	No
AVIF	AV1 Image File Format	No	**Yes**	**Yes**
BAG	Bathymetry Attributed Grid	**Yes**	**Yes**	**Yes**
BASISU	Basis Universal	No	**Yes**	No
BMP	Microsoft Windows Device Independent Bitmap	**Yes**	**Yes**	**Yes**
BSB	Maptech/NOAA BSB Nautical Chart Format	No	No	**Yes**
BT	VTP .bt Binary Terrain Format	**Yes**	**Yes**	**Yes**
BYN	Natural Resources Canada's Geoid file format (.byn)	No	No	**Yes**
CAD	AutoCAD DWG raster layer	No	No	**Yes**
CALS	CALS Type 1	No	**Yes**	No
CEOS	CEOS Image	No	No	No
COASP	DRDC COASP SAR Processor Raster	No	No	No
COG	Cloud Optimized GeoTIFF generator	**Yes**	**Yes**	**Yes**
COSAR	TerraSAR-X Complex SAR Data Product	No	No	No
CPG	Convair PolGASP data	No	No	**Yes**
CPHD	Compensated Phase History Data	No	No	No
CTG	USGS LULC Composite Theme Grid	No	No	**Yes**
DAAS	DAAS (Airbus DS Intelligence Data As A Service driver)	No	No	**Yes**
DDS	DirectDraw Surface	No	**Yes**	No
DERIVED	Derived subdatasets driver	No	No	No
DIMAP	Spot DIMAP	No	No	**Yes**
DOQ1	First Generation USGS DOQ	No	No	**Yes**
DOQ2	New Labelled USGS DOQ	No	No	**Yes**
DTED	Military Elevation Data	No	**Yes**	**Yes**
E57	E57: ASTM E57 3D file format (image part)	No	No	No
ECRGTOC	ECRG Table Of Contents (TOC.xml)	No	No	**Yes**
ECW	Enhanced Compressed Wavelets (.ecw)	**Yes**	**Yes**	**Yes**
EEDAI	Google Earth Engine Data API Image	No	No	**Yes**
EHdr	ESRI .hdr Labelled	**Yes**	**Yes**	**Yes**
EIR	Erdas Imagine Raw	No	No	**Yes**
ENVI	ENVI .hdr Labelled Raster	**Yes**	**Yes**	**Yes**
ERS	ERMapper .ERS	**Yes**	**Yes**	**Yes**
ESAT	Envisat Image Product	No	No	No
ESRIC	Esri Compact Cache	No	No	**Yes**

Table 1 – continued from

Short Name	Long Name	Creation (1)	Copy (2)	Georef
EXR	Extended Dynamic Range Image File Format	**Yes**	**Yes**	**Yes**
FAST	EOSAT FAST Format	No	No	**Yes**
FITS	Flexible Image Transport System	**Yes**	**Yes**	**Yes**
GDALG	GDALG: GDAL Streamed Algorithm	No	No	**Yes**
GenBin	Generic Binary (.hdr labelled)	No	No	No
GeoRaster	Oracle Spatial GeoRaster	**Yes**	**Yes**	**Yes**
GFF	Sandia National Laboratories GSAT File Format	No	No	No
GIF	Graphics Interchange Format	No	**Yes**	No
GPKG	GeoPackage raster	**Yes**	**Yes**	**Yes**
GRASS	GRASS Raster Format	No	No	No
GRASSASCIIGrid	GRASS ASCII Grid	No	No	**Yes**
GRIB	WMO General Regularly-distributed Information in Binary form	No	**Yes**	**Yes**
GS7BG	Golden Software Surfer 7 Binary Grid File Format	**Yes**	**Yes**	**Yes**
GSAG	Golden Software ASCII Grid File Format	No	**Yes**	**Yes**
GSBG	Golden Software Binary Grid File Format	**Yes**	**Yes**	**Yes**
GSC	GSC Geogrid	No	No	No
GTA	Generic Tagged Arrays	No	**Yes**	**Yes**
GTI	GDAL Raster Tile Index	No	No	No
GTiff	GeoTIFF File Format	**Yes**	**Yes**	**Yes**
GTX	GTX — NOAA Vertical Datum Grid Shift	**Yes**	**Yes**	**Yes**
GXF	Grid eXchange File	No	No	**Yes**
HDF4	Hierarchical Data Format Release 4 (HDF4)	**Yes**	**Yes**	**Yes**
HDF5	Hierarchical Data Format Release 5 (HDF5)	No	No	**Yes**
HEIF	ISO/IEC 23008-12 High Efficiency Image File Format	No	**Yes**	**Yes**
HF2	HF2/HFZ heightfield raster	No	**Yes**	**Yes**
HFA	Erdas Imagine .img	**Yes**	**Yes**	**Yes**
ILWIS	Raster Map	**Yes**	**Yes**	**Yes**
IRIS	Vaisala's weather radar software format	No	No	**Yes**
ISCE	ISCE	**Yes**	**Yes**	**Yes**
ISG	International Service for the Geoid	No	No	**Yes**
ISIS2	USGS Astrogeology ISIS Cube (Version 2)	No	No	**Yes**
ISIS3	USGS Astrogeology ISIS Cube (Version 3)	**Yes**	**Yes**	**Yes**
JAXAPALSAR	JAXA PALSAR Processed Products	No	No	**Yes**
JDEM	Japanese DEM (.mem)	No	No	**Yes**
JP2ECW	ERDAS JPEG2000 (.jp2)	**Yes**	**Yes**	**Yes**
JP2Grok	JPEG2000 driver based on Grok library	No	**Yes**	**Yes**
JP2KAK	JPEG 2000 (based on Kakadu SDK)	No	**Yes**	**Yes**
JP2MrSID	JPEG2000 via MrSID SDK	No	**Yes**	**Yes**
JP2OpenJPEG	JPEG2000 driver based on OpenJPEG library	No	**Yes**	**Yes**
JPEG	JPEG JFIF File Format	No	**Yes**	**Yes**
JPEGXL	JPEG-XL File Format	No	**Yes**	**Yes**
JPIPKAK	JPIP Streaming	No	No	**Yes**
KEA	KEA	**Yes**	**Yes**	**Yes**
KMLSuperoverlay	KMLSuperoverlay	No	**Yes**	**Yes**
KRO	KOLOR Raw format	**Yes**	**Yes**	No
KTX2	KTX2	No	**Yes**	No
L1B	NOAA Polar Orbiter Level 1b Data Set (AVHRR)	No	No	**Yes**
LAN	Erdas 7.x .LAN and .GIS	No	No	**Yes**
LCP	FARSITE v.4 LCP Format	No	**Yes**	**Yes**
Leveller	Daylon Leveller Heightfield	**Yes**	**Yes**	**Yes**

Table 1 – continued from

Short Name	Long Name	Creation (1)	Copy (2)	Georef
LIBERTIFF	GeoTIFF File Format	No	No	**Yes**
LOSLAS	NADCON .los/.las Datum Grid Shift	No	No	**Yes**
MAP	OziExplorer .MAP	No	No	**Yes**
MBTiles	MBTiles	**Yes**	**Yes**	**Yes**
MEM	In Memory datasets	**Yes**	**Yes**	**Yes**
MFF2	Vexcel MFF2 Image	No	No	**Yes**
MFF	Vexcel MFF Raster	No	No	**Yes**
MiraMonRaster	MiraMon Raster	No	**Yes**	**Yes**
MRF	Meta Raster Format	**Yes**	**Yes**	**Yes**
MrSID	Multi-resolution Seamless Image Database	No	No	**Yes**
MSG	Meteosat Second Generation	No	No	**Yes**
MSGN	Meteosat Second Generation (MSG) Native Archive Format (.nat)	No	No	**Yes**
NDF	NLAPS Data Format	No	No	**Yes**
netCDF	NetCDF: Network Common Data Form	**Yes**	**Yes**	**Yes**
NGSGEOID	NOAA NGS Geoid Height Grids	No	No	**Yes**
NGW	NextGIS Web	No	No	**Yes**
NITF	National Imagery Transmission Format (also CIB, CADRG, ECRG, HRE)	**Yes**	**Yes**	**Yes**
NOAA_B	NOAA GEOCON/NADCON5 .b format	No	No	**Yes**
NSIDCbin	National Snow and Ice Data Centre Sea Ice Concentrations	No	No	**Yes**
NTv2	NTv2 Datum Grid Shift	No	No	**Yes**
NWT_GRD	Northwood/Vertical Mapper File Format	**Yes**	**Yes**	**Yes**
OGCAPI	OGC API Tiles / Maps / Coverage	No	No	**Yes**
OpenFileGDB	ESRI File Geodatabase raster (OpenFileGDB)	No	No	**Yes**
PAux	PCI .aux Labelled Raw Format	No	No	No
PCIDSK	PCI Geomatics Database File	**Yes**	**Yes**	**Yes**
PCRaster	PCRaster raster file format	**Yes**	**Yes**	**Yes**
PDF	Geospatial PDF	No	**Yes**	**Yes**
PDS4	NASA Planetary Data System (Version 4)	**Yes**	**Yes**	**Yes**
PDS	Planetary Data System v3	No	No	**Yes**
PLMosaic	PLMosaic (Planet Labs Mosaics API)	No	No	**Yes**
PMTiles	PMTiles	No	**Yes**	**Yes**
PNG	Portable Network Graphics	No	**Yes**	**Yes**
PNM	Netpbm (.pgm, .ppm)	**Yes**	**Yes**	No
PostGISRaster	PostGIS Raster driver	No	**Yes**	**Yes**
PRF	PHOTOMOD Raster File	No	No	No
RCM	RADARSAT Constellation Mission Product	No	No	**Yes**
RIK	Swedish Grid Maps	No	No	**Yes**
RMF	Raster Matrix Format	**Yes**	**Yes**	**Yes**
ROI_PAC	ROI_PAC	**Yes**	**Yes**	**Yes**
RPFTOC	Raster Product Format/RPF (a.toc)	No	No	**Yes**
RRASTER	R Raster	**Yes**	**Yes**	**Yes**
RS2	RadarSat 2 XML Product	No	No	**Yes**
RST	Idrisi Raster Format	**Yes**	**Yes**	**Yes**
S102	S-102 Bathymetric Surface Product	No	**Yes**	**Yes**
S104	S-104 Water Level Information for Surface Navigation Product	No	**Yes**	**Yes**
S111	S-111 Surface Currents Product	No	**Yes**	**Yes**
SAFE	Sentinel-1 SAFE XML Product	No	No	**Yes**
SAGA	SAGA GIS Binary Grid File Format	**Yes**	**Yes**	**Yes**
SAR_CEOS	CEOS SAR Image	No	No	**Yes**
SENTINEL2	Sentinel-2 Products	No	No	**Yes**

Table 1 – continued from

Short Name	Long Name	Creation (1)	Copy (2)	Georef
SIGDEM	Scaled Integer Gridded DEM	No	**Yes**	**Yes**
SNAP_TIFF	Sentinel Application Processing GeoTIFF	No	No	**Yes**
SNODAS	Snow Data Assimilation System	No	No	**Yes**
SQLite	Rasters in SQLite DB	No	**Yes**	**Yes**
SRP	Standard Product Format (ASRP/USRP) (.gen)	No	No	**Yes**
SRTMHGT	SRTM HGT Format	No	**Yes**	**Yes**
STACIT	Spatio-Temporal Asset Catalog Items	No	No	No
STACTA	Spatio-Temporal Asset Catalog Tiled Assets	No	No	No
Terragen	Terragen™ Terrain File	**Yes**	**Yes**	**Yes**
TGA	TARGA Image File Format	No	No	No
TIL	EarthWatch/DigitalGlobe .TIL	No	No	**Yes**
TileDB	TileDB raster	**Yes**	**Yes**	**Yes**
TSX	TerraSAR-X Product	No	No	**Yes**
USGSDEM	USGS ASCII DEM (and CDED)	No	No	**Yes**
VICAR	VICAR	**Yes**	**Yes**	**Yes**
VRT	GDAL Virtual Format	**Yes**	**Yes**	**Yes**
WCS	OGC Web Coverage Service	No	No	**Yes**
WEBP	WEBP	No	**Yes**	No
WMS	Web Map Services	No	**Yes**	**Yes**
WMTS	OGC Web Map Tile Service	No	No	**Yes**
XYZ	ASCII Gridded XYZ	No	**Yes**	**Yes**
Zarr	Zarr	**Yes**	**Yes**	**Yes**
ZMAP	ZMap Plus Grid	No	**Yes**	**Yes**

- (1): Creation refers to implementing `GDALCreate()`.
- (2): Copy refers to implementing `GDALCreateCopy()`.

Note

The following drivers have been removed in GDAL 3.5: BPG, E00GRID, EPSILON, IGNFHeightASCIIGrid, NTv1

The following drivers have been removed in GDAL 3.11: BLX, CTable2, ELAS, FIT, JP2Lura, OZI, Rasterlite (v1), R object data store (.rda), RDB, SDTS, SGI, XPM

Write support for the following formats has been removed in GDAL 3.11: ADRG, BYN, LAN, MFF, MFF2/HKV, ISIS2, PAux, USGSDEM

6.1 AAIGrid -- Arc/Info ASCII Grid

Driver short name

AAIGrid

Driver built-in by default

This driver is built-in by default

Supported for read and write access, including reading of an affine georeferencing transform and some projections. This format is the ASCII interchange format for Arc/Info Grid, and takes the form of an ASCII file, plus sometimes an associated .prj file. It is normally produced with the Arc/Info ASCIIGRID command.

The projections support (read if a *.prj file is available) is quite limited. Additional sample .prj files may be sent to the maintainer, warmerdam@pobox.com.

The NODATA value for the grid read is also preserved when available in the same format as the band data.

The following configuration options are supported:

- **AAIGRID_DATATYPE=[Int32/Float32/Float64]:** By default, the datatype returned for AAIGRID datasets by GDAL is autodetected, and set to Float32 for grid with floating point values or Int32 otherwise. This is done by analysing the format of the NODATA value and, if needed, the data of the grid. You can explicitly specify the datatype by setting the AAIGRID_DATATYPE configuration option.
- **FORCE_CELLSIZE=[YES/NO]:** If pixels being written are not square (the width and height of a pixel in georeferenced units differ) then DX and DY parameters will be output instead of CELLSIZE. Such files can be used in Golden Surfer, but not most other ascii grid reading programs. For force the X pixel size to be used as CELLSIZE use the FORCE_CELLSIZE=YES creation option or resample the input to have square pixels.
- **DECIMAL_PRECISION=<integer>:** When writing floating-point values, the driver uses the "%.20g" format pattern as a default. You can consult a reference manual for printf to have an idea of the exact behavior of this ;-). You can alternatively specify the number of decimal places with the DECIMAL_PRECISION creation option. For example, DECIMAL_PRECISION=3 will output numbers with 3 decimal places(using %lf format). Another option is SIGNIFICANT_DIGITS=3, which will output 3 significant digits (using %g format).

The *AIG -- Arc/Info Binary Grid* driver is also available for Arc/Info Binary Grid format.

NOTE: Implemented as aaigriddataset.cpp.

6.1.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.2 ACE2 -- ACE2

Driver short name

ACE2

Driver built-in by default

This driver is built-in by default

This is a convenience driver to read ACE2 DEMs. Those files contain raw binary data. The georeferencing is entirely determined by the filename. Quality, source and confidence layers are of Int16 type, whereas elevation data is returned as Float32.

ACE2 product overview

NOTE: Implemented as ace2dataset.cpp.

6.2.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.3 ADRG -- ADRG/ARC Digitized Raster Graphics (.gen/.thf)

Driver short name

ADRG

Driver built-in by default

This driver is built-in by default

Supported by GDAL for read access.

An ADRG dataset is made of several files. The file recognised by GDAL is the General Information File (.GEN). GDAL will also need the image file (.IMG), where the actual data is.

The Transmission Header File (.THF) can also be used as an input to GDAL. If the THF references more than one image, GDAL will report the images it is composed of as subdatasets. If the THF references just one image, GDAL will open it directly.

Overviews, legends and insets are not used. Polar zones (ARC zone 9 and 18) are not supported (due to the lack of test data).

See also : the ADRG specification (MIL-A-89007)

6.3.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.4 AIG -- Arc/Info Binary Grid

Driver short name

AIG

Driver built-in by default

This driver is built-in by default

Supported by GDAL for read access. This format is the internal binary format for Arc/Info Grid, and takes the form of a coverage level directory in an Arc/Info database. To open the coverage select the coverage directory, or an .adf file (such as hdr.adf) from within it. If the directory does not contain file(s) with names like w001001.adf then it is not a grid coverage.

Support includes reading of an affine georeferencing transform, some projections, and a color table (.clr) if available.

This driver is implemented based on a reverse engineering of the format. See the *Arc/Info Binary Grid Format* for more details.

The projections support (read if a prj.adf file is available) is quite limited. Additional sample prj.adf files may be sent to the maintainer, warmerdam@pobox.com.

NOTE: Implemented as aigdataset.cpp.

6.4.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.4.1.1 Arc/Info Binary Grid Format

by Frank Warmerdam (warmerdam@pobox.com)

The Arc/Info Binary Grid format is the internal working format of the Arc/Info Grid product. It is also usable and creatable within the spatial analyst component of ArcView. It is a tiled (blocked) format with run length compression capable of holding raster data of up to 4 byte integers or 4 byte floating data.

This format should not be confused with the Arc/Info ASCII Grid format which is the interchange format for grids. Files can be converted between binary and ASCII format with the GRIDASCII and ASCIIGRID commands in Arc/Info. This format is also different than the flat binary raster output of the GRIDFLOAT command. The Arc/Info binary float, and ASCII formats are also accessible from within ArcView.

This format should also not be confused with what I know as ESRI BIL format. This is really a standard ESRI way of creating a header file (.HDR) describing the data layout a binary raster file containing raster data.

Version

I am not sure yet how the versions work for grid files. I have been working primarily with grid files generated by ArcView 3.x, and its associated gridio API. The hdr.adf files I have examined start with the string **GRID1.2** for what that's worth. Certainly the file naming conventions seem to follow the Arc/Info 7.x conventions rather than that of earlier versions.

File Set

A grid coverage actually consists of a number of files. A grid normally lives in its own directory named after the grid. For instance, the grid **nwgrd1** lives in the directory **nwgrd1**, and has the following component files:

```
-rwxr--r--   1 warmerda users          32 Jan 22 16:07 nwgrd1/dblbnd.adf
-rwxr--r--   1 warmerda users         308 Jan 22 16:07 nwgrd1/hdr.adf
-rwxr--r--   1 warmerda users          32 Jan 22 16:07 nwgrd1/sta.adf
-rwxr--r--   1 warmerda users        2048 Jan 22 16:07 nwgrd1/vat.adf
-rwxr--r--   1 warmerda users      187228 Jan 22 16:07 nwgrd1/w001001.adf
-rwxr--r--   1 warmerda users        6132 Jan 22 16:07 nwgrd1/w001001x.adf
```

Sometimes datasets will also include a prj.adf files containing the projection definition in the usual ESRI format. Grids also normally have associated tables in the info directory. This is beyond the scope of my discussion for now.

The files have the following roles:

- *dblbnd.adf*: Contains the bounds (LLX, LLY, URX, URY) of the portion of utilized portion of the grid.
- *hdr.adf*: This is the header, and contains information on the tile sizes, and number of tiles in the dataset. It also contains assorted other information I have yet to identify.
- *sta.adf*: This contains raster statistics. In particular, the raster min, max, mean and standard deviation.
- **vat.adf**: This relates to the value attribute table. This is the table corresponding integer raster values with a set of attributes. I presume it is really just a pointer into info in a manner similar to the pat.adf file in a vector coverage, but I haven't investigated yet.
- *w001001.adf*: This is the file containing the actual raster data.
- *w001001x.adf*: This is an index file containing pointers to each of the tiles in the w001001.adf raster file.

---

dblbnd.adf - Georef Bounds

Fields:

Start Byte

# of Bytes

Format

Name

Description

0

8

MSB double

D_LLX

Lower left X (easting) of the grid. Generally -0.5 for an ungeoreferenced grid.

8

8

MSB double

D_LLY

Lower left Y (northing) of the grid. Generally -0.5 for an ungeoreferenced grid.

16

8

MSB double

D_URX

Upper right X (easting) of the grid. Generally #Pixels-0.5 for an ungeoreferenced grid.

24

8

MSB double

D_URY

Upper right Y (northing) of the grid. Generally #Lines-0.5 for an ungeoreferenced grid.

This file is always 32 bytes long. The bounds apply to the portion of the grid that is in use, not the whole thing.

---

w001001x.adf - Tile Index

This is a binary dump of the first 320 bytes of a w001001x.adf file.

```
  0: 0000270A FFFFFC14 00000000 00000000 ~~'~~~~~~~~~~~~
 16: 00000000 00000000 00000BFA 00000000 ~~~~~~~~~~~~~~~~
 32: 00000000 00000000 00000000 00000000 ~~~~~~~~~~~~~~~~
 48: 00000000 00000000 00000000 00000000 ~~~~~~~~~~~~~~~~
 64: 00000000 00000000 00000000 00000000 ~~~~~~~~~~~~~~~~
 80: 00000000 00000000 00000000 00000000 ~~~~~~~~~~~~~~~~
 96: 00000000 00000032 00000202 00000235 ~~~~~~~2~~~~~~~5
112: 000001D4 0000040A 00000000 0000040B ~~~~~~~~~~~~~~~~
128: 00000000 0000040C 00000000 0000040D ~~~~~~~~~~~~~~~~
144: 00000000 0000040E 00000000 0000040F ~~~~~~~~~~~~~~~~
160: 00000000 00000410 00000202 00000613 ~~~~~~~~~~~~~~~~
```

(continues on next page)

(continued from previous page)

```
176: 000001D4 000007E8 00000000 000007E9 ~~~~~~~~~~~~~~~~
192: 00000000 000007EA 00000000 000007EB ~~~~~~~~~~~~~~~~
208: 00000000 000007EC 00000000 000007ED ~~~~~~~~~~~~~~~~
224: 00000000 000007EE 00000202 000009F1 ~~~~~~~~~~~~~~~~
240: 000001D4 00000BC6 00000000 00000BC7 ~~~~~~~~~~~~~~~~
256: 00000000 00000BC8 00000000 00000BC9 ~~~~~~~~~~~~~~~~
272: 00000000 00000BCA 00000000 00000BCB ~~~~~~~~~~~~~~~~
288: 00000000 00000BCC 00000202 00000DCF ~~~~~~~~~~~~~~~~
304: 000001D4 00000FA4 00000000 00000FA5 ~~~~~~~~~~~~~~~~
```

Fields:

Start Byte

# of Bytes

Format

Description

0

8

Magic Number (always hex 00 00 27 0A FF FF ** **, usually ending in FC 14, FB F8 or FC 08).

8

16

zero fill

24

4

MSB Int32

Size of whole file in shorts (multiply by two to get file size in bytes).

28

72

zero fill

100 + **t***8

4

MSB Int32

Offset to tile **t** in w001001.adf measured in two byte shorts.

104 + **t***8

4

MSB Int32

Size of tile **t** in 2 byte shorts.

sta.adf - Raster Statistics

Fields:

Start Byte

# of Bytes

Format

Name

Description

0

8

MSB double

SMin

Minimum value of a raster cell in this grid.

8

8

MSB double

SMax

Maximum value of a raster cell in this grid.

16

8

MSB double

SMean

Mean value of a raster cells in this grid.

24

8

MSB double

SStdDev

Standard deviation of raster cells in this grid.

This file is always 32 bytes long.

---

w001001.adf - Raster Data

This is a binary dump of the first 320 bytes of a w001001.adf file.

```
 0: 0000270A FFFFFC14 00000000 00000000 ~~'~~~~~~~~~~~~~
16: 00000000 00000000 00016DAE 00000000 ~~~~~~~~~~m~~~~~
32: 00000000 00000000 00000000 00000000 ~~~~~~~~~~~~~~~~
48: 00000000 00000000 00000000 00000000 ~~~~~~~~~~~~~~~~
64: 00000000 00000000 00000000 00000000 ~~~~~~~~~~~~~~~~
```

(continues on next page)

(continued from previous page)

```
 80: 00000000 00000000 00000000 00000000 ~~~~~~~~~~~~~~~~
 96: 00000000 02020800 00373D42 5C5A4D31 ~~~~~~~~~~7=B\ZM1
112: 200A0108 0E1D4F89 9C9A9392 8C7E6653  ~~~~~O~~~~~~~fS
128: 5151596D 83919290 868A8B87 807A7A7B QQYm~~~~~~~~~zz{
144: 7C7A766F 64481D00 0406305F 6B6C6A5B |zvodH~~~~0_klj[
160: 5D53513C 2D2D2732 24293F54 40354C55 ]SQ<--'2$)?T@5LU
176: 67686258 514E4943 5859534A 41394D70 ghbXQNICXYSJA9Mp
192: 75665659 66625A63 737A848E 9090979F ufVYfbZcsz~~~~~~
208: 9F908C8F 8F96998E 8778685B 53536274 ~~~~~~~~~xh[SSbt
224: 747B838A 8A8C8F92 8D979B94 8C8D9294 t{~~~~~~~~~~~~~~
240: 8D8D8D8D 8C8B8989 8B8E908F 8E8E9092 ~~~~~~~~~~~~~~~~
256: 90929394 989C9891 92939698 9B9B9C9C ~~~~~~~~~~~~~~~~
272: 8E8E8F8F 8E8E8F90 898E918F 8B8A8E93 ~~~~~~~~~~~~~~~~
288: 8B8D9093 94918C86 838DA1BC B7CEC9B0 ~~~~~~~~~~~~~~~~
304: D4B0BB96 A0929E99 9797999B 9D9C9C9B ~~~~~~~~~~~~~~~~
```

Fields:

Start Byte

# of Bytes

Format

Name

Description

0

8

RMagic

Magic Number (always hex 00 00 27 0A FF FF ** **, usually ending in FC 14, FB F8 or FC 08).

8

16

zero fill

24

4

MSB Int32

RFileSize

Size of whole file in shorts (multiply by two to get file size in bytes).

28

72

zero fill

100, ...

2

MSB Int16

RTileSize

Size of this tiles data measured in shorts. This matches the size in the index file, and does not include the tile size itself. The next tile starts **2*n+2** bytes after the start of this tile, where **n** is the value of this field.

102, ...

1

byte

RTileType

Tile type code indicating the organization of the following data (integer coverages only).

103, ...

1

byte

RMinSize

Number of bytes following to form the minimum value for the tile (integer coverages only).

104, ...

(RMinSize bytes)

MSB Int (var size)

RMin

The minimum value pixels for this tile. This number is added to the pixel values for each pixel in this tile (integer coverages only). I must stress that if RMinSize is less than 4 this is still a signed quantity. For instance, if RMinSize is 2, the value is 65536 - byte0*256 - byte1 if byte0 is > 127.

104+RMinSize, ...

RTileSize*2 - 3 - RMinSize

variable

RTileData

The data for this tile. Format varies according to RTileType for integer coverages.

The fields RTileSize, RTileType, RMinSize, RMin, and RTileData occur in the file for each tile of data present. They are usually packed one after the other, but this isn't necessarily guaranteed. The index file (w001001x.adf) should be used to establish the tile locations. Note that tiles that appear in the index file with a size of zero will appear as just two bytes (zeros) for the RTileSize for that tile.

Raster Size

The size of a the grid isn't as easy to deduce as one might expect. The hdr.adf file contains the HTilesPerRow, HTilesPerColumn, HTileXSize, and HTileYSize fields which imply a particular raster space. However, it seems that this is created much larger than necessary to hold the users raster data. I have created 3x1 rasters which resulted in the standard 8x512 tiles of 256x4 pixels each.

It seems that the user portion of the raster has to be computed based on the georeferenced bounds in the dblbnd.adf file (assumed to be anchored at the top left of the total raster space), and the HPixelSizeX, and HPixelSizeY fields from hdr.adf.

#Pixels = (D_URX - D_LRX) / HPixelSizeX

#Lines = (D_URY - D_LRY) / HPixelSizeY

Based on this number of pixels and lines, it is possible to establish what portion out of the top left of the raster is really *of interest*. All regions outside this appear to empty tiles, or filled with no data markers.

RTileType/RTileData

Each tile contains HBlockXSize * HBlockYSize pixels of data. For floating point and uncompressed integer files the data is just the tile size (in two bytes) followed by the pixel data as 4 byte MSB order IEEE floating point words. For compressed integer tiles it is necessary to interpret the RTileType to establish the details of the tile organization:

RTileType = 0x00 (constant block)

All pixels take the value of the RMin. Data is ignored. It appears there is sometimes a bit of meaningless data (up to four bytes) in the block.

RTileType = 0x01 (raw 1bit data)

One full tile worth of data pixel values follows the RMin field, with 1bit per pixel.

RTileType = 0x04 (raw 4bit data)

One full tiles worth of data pixel values follows the RMin field, with 4 bits per pixel. The high order four bits of a byte comes before the low order four bits.

RTileType = 0x08 (raw byte data)

One full tiles worth of data pixel values (one byte per pixel) follows the RMin field.

RTileType = 0x10 (raw 16bit data)

One full tiles worth of data pixel values follows the RMin field, with 16 bits per pixel (MSB).

RTileType = 0x20 (raw 32bit data)

One full tiles worth of data pixel values follows the RMin field, with 32 bits per pixel (MSB).

RTileType = 0xCF (16 bit literal runs/nodata runs)

The data is organized in a series of runs. Each run starts with a marker which should be interpreted as:

- **Marker < 128**: The marker is followed by **Marker** pixels of literal data with two MSB bytes per pixel.
- **Marker > 127**: The marker indicates that **256-Marker** pixels of *no data* pixels should be put into the output stream. No data (other than the next marker) follows this marker.

RTileType = 0xD7 (literal runs/nodata runs)

The data is organized in a series of runs. Each run starts with a marker which should be interpreted as:

- **Marker < 128**: The marker is followed by **Marker** pixels of literal data with one byte per pixel.
- **Marker > 127**: The marker indicates that **256-Marker** pixels of *no data* pixels should be put into the output stream. No data (other than the next marker) follows this marker.

RTileType = 0xDF (RMin runs/nodata runs)

The data is organized in a series of runs. Each run starts with a marker which should be interpreted as:

- **Marker < 128**: The marker is followed by **Marker** pixels of literal data with one byte per pixel.
- **Marker > 127**: The marker indicates that **256-Marker** pixels of *no data* pixels should be put into the output stream. No data (other than the next marker) follows this marker.

This is similar to 0xD7, except that the data size is zero bytes instead of 1, so only RMin values are inserted into the output stream.

RTileType = 0xE0 (run length encoded 32bit)

The data is organized in a series of runs. Each run starts with a marker which should be interpreted as a **count**. The four bytes following the count should be interpreted as an MSB Int32 **value**. They indicate that **count** pixels of **value** should be inserted into the output stream.

RTileType = 0xF0 (run length encoded 16bit)

The data is organized in a series of runs. Each run starts with a marker which should be interpreted as a **count**. The two bytes following the count should be interpreted as an MSB Int16 **value**. They indicate that **count** pixels of **value** should be inserted into the output stream.

RTileType = 0xFC/0xF8 (run length encoded 8bit)

The data is organized in a series of runs. Each run starts with a marker which should be interpreted as a **count**. The following byte is the **value**. They indicate that **count** pixels of **value** should be inserted into the output stream.

The interpretation is the same for 0xFC, and 0xF8. I believe that 0xFC has a lower dynamic (2 bit) range than 0xF8 (4 or 8 bit).

RTileType = 0xFF (RMin CCITT RLE 1Bit)

The data stream for this file is CCITT RLE (G1 fax) compressed. The format is complex but source is provided with the sample program (derived from libtiff) for reading it. The result of uncompressing is 1bit data so which the RMin value should be added.

---

hdr.adf - Header

This is a binary dump of the first 308 bytes of a hdr.adf file.

```
  0: 47524944 312E3200 00000000 FFFFFFFF GRID1.2~~~~~~~~~
 16: 00000001 00000000 0000164E 3F800000 ~~~~~~~~~~~N?~~~
 32: 00000F00 F6180000 90060000 3603D601 ~~~~~~~~~~~~6~~~
 48: 6403E301 01000000 7620F808 43012B03 d~~~~~~~v ~~C~+~
 64: D6019903 E3012B03 D6019903 E301F7BF ~~~~~~+~~~~~~~~~
 80: 00007406 6E1FC2A4 7A370D00 0B004200 ~~t~n~~~z7~~~~B~
 96: 4E1654A4 00000000 00000000 00000000 N~T~~~~~~~~~~~~
112: 34A5A89D FF0414A5 A70F0002 00000000 4~~~~~~~~~~~~~~~
128: 00000000 3C0B5F06 A8C05F06 08005AC0 ~~~~<~_~~~_~~~Z~
144: 0A00E101 36035AC0 72085F06 FAA42F3C ~~~~6~Z~r~_~~~/<
160: 0A001667 02000E00 A80B0200 08370200 ~~~g~~~~~~~~~7~~
176: 0CA00200 9C0B0200 04370200 36A0E436 ~~~~~~~~~7~~6~~6
```

(continues on next page)

(continued from previous page)

```
192: 84000000 36A00200 5F063EA5 0883FF04 ~~~~6~~~_~>~~~~~
208: 00008400 00000010 BD810200 5F010000 ~~~~~~~~~~~~_~~~
224: 670E0000 5F01560E 4C4F0001 84008CA5 g~~~_~V~LO~~~~~~
240: 28008F01 1000E00A 6628F7BF 4076FF04 (~~~~~~~f(~~@v~~
256: 3FF00000 00000000 3FF00000 00000000 ?~~~~~~~?~~~~~~~
272: C08FFC00 00000000 C0A1BF00 00000000 ~~~~~~~~~~~~~~~~
288: 00000008 00000200 00000100 00000001 ~~~~~~~~~~~~~~~~
304: 00000004                            ~~~~
```

Fields:

Start Byte

# of Bytes

Format

Name

Description

0

8

Char

HMagic

Magic Number - always "GRID1.20"

8

8

assorted data, I don't know the purpose.

16

4

MSB Int32

HCellType

1 = int cover, 2 = float cover.

20

4

MSB Int32

CompFlag

0 = compressed, 1 = uncompressed

24

232

assorted data, I don't know the purpose.

256

8

MSB Double

HPixelSizeX

Width of a pixel in georeferenced coordinates. Generally 1.0 for ungeoreferenced rasters.

264

8

MSB Double

HPixelSizeY

Height of a pixel in georeferenced coordinates. Generally 1.0 for ungeoreferenced rasters.

272

8

MSB Double

XRef

dfLLX-(nBlocksPerRow*nBlockXSize*dfCellSizeX)/2.0

280

8

MSB Double

YRef

dfURY-(3*nBlocksPerColumn*nBlockYSize*dfCellSizeY)/2.0

288

4

MSB Int32

HTilesPerRow

The width of the file in tiles (often 8 for files of less than 2K in width).

292

4

MSB Int32

HTilesPerColumn

The height of the file in tiles. Note this may be much more than the number of tiles actually represented in the index file.

296

4

MSB Int32

HTileXSize

The width of a file in pixels. Normally 256.

300

4

MSB Int32

Unknown, usually 1.

304

4

MSB Int32

HTileYSize

Height of a tile in pixels, usually 4.

Acknowledgements

I would like to thank Geosoft Inc. for partial funding of my research into this format. I would also like to thank:

- Kenneth R. McVay for providing the statistics file format.
- Noureddine Farah of ThinkSpace who dug up lots of datasets that caused problems.
- Luciano Fonseca who worked out RTileType 0x01.
- Martin Manningham of Global Geomatics for additional problem sample files.
- Harry Anderson of EDX Engineering, for showing me that floating point tiles don't have RTileType.
- Ian Turton for supplying a sample files demonstrating the need to be careful with the sign of "short" RMin values.
- Duncan Chaundy at PCI for poking hard till I finally deduced 0xFF tiles.
- Stephen Cheeseman of GeoSoft for yet more problem files.
- Geoffrey Williams for a files demonstrating tile type 0x20.

6.5 AIRSAR -- AIRSAR Polarimetric Format

Driver short name

AIRSAR

Driver built-in by default

This driver is built-in by default

Most variants of the AIRSAR Polarimetric Format produced by the AIRSAR Integrated Processor are supported for reading by GDAL. AIRSAR products normally include various associated data files, but only the imagery data themselves is supported. Normally these are named *mission*_l.dat (L-Band) or *mission*_c.dat (C-Band).

AIRSAR format contains a polarimetric image in compressed stokes matrix form. Internally GDAL decompresses the data into a stokes matrix, and then converts that form into a covariance matrix. The returned six bands are the six values needed to define the 3x3 Hermitian covariance matrix. The convention used to represent the covariance matrix in terms of the scattering matrix elements HH, HV (=VH), and VV is indicated below. Note that the non-diagonal elements of the matrix are complex values, while the diagonal values are real (though represented as complex bands).

- Band 1: Covariance_11 (Float32) = HH*conj(HH)

- Band 2: Covariance_12 (CFloat32) = sqrt(2)*HH*conj(HV)
- Band 3: Covariance_13 (CFloat32) = HH*conj(VV)
- Band 4: Covariance_22 (Float32) = 2*HV*conj(HV)
- Band 5: Covariance_23 (CFloat32) = sqrt(2)*HV*conj(VV)
- Band 6: Covariance_33 (Float32) = VV*conj(VV)

The identities of the bands are also reflected in metadata and in the band descriptions.

The AIRSAR product format includes (potentially) several headers of information. This information is captured and represented as metadata on the file as a whole. Information items from the main header are prefixed with "MH_", items from the parameter header are prefixed with "PH_" and information from the calibration header are prefixed with "CH_". The metadata item names are derived automatically from the names of the fields within the header itself.

No effort is made to read files associated with the AIRSAR product such as *mission*_l.mocomp, *mission*_meta.airsar or *mission*_meta.podaac.

6.5.1 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.5.2 See Also

- AIRSAR Data Format

6.6 AVIF -- AV1 Image File Format

Added in version 3.10.

Driver short name

AVIF

Build dependencies

libavif

AV1 Image File Format (AVIF) is an open, royalty-free image file format specification for storing images or image sequences compressed with AV1 in the HEIF container format.

It supports 8-bit, 10-bit and 12-bit images, single-bland, single-band and alpha channel, RGB and RGBA.

Files containing animations (several images) will be exposed as GDAL subdatasets.

Compression and decompression is done on entire images, so the driver can not handle arbitrary sizes images.

Note: read-only support for AVIF files is also available through the *HEIF / HEIC -- ISO/IEC 23008-12 High Efficiency Image File Format* driver if the AVIF driver is not available, and if libheif has been compiled with an AV1 compatible decoder.

6.6.1 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Georeferencing

This driver supports georeferencing

6.6.2 Color Profile Metadata

GDAL can deal with the following color profile metadata in the COLOR_PROFILE domain:

- SOURCE_ICC_PROFILE (Base64 encoded ICC profile embedded in file.)

6.6.3 Georeferencing

AVIF provides experimental read-only support for GeoHEIF internal referencing, based on the draft OGC 24-038 design. There is no support for writing at this time.

This requires at least libavif 1.2.0 (currently unreleased).

6.6.4 Creation options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following creation options are supported:

- **CODEC=[AUTO/AOM/RAV1E/SVT]:** Defaults to `AUTO`. Compression library to use. Choices available depend on how libavif has been built.
- **QUALITY=[0-100]:** Defaults to `60`. Quality for non-alpha channels. 0 is the lowest quality, 100 is for lossless encoding. Default is 60.
- **QUALITY_ALPHA=[0-100]:** Defaults to `100`. Quality for alpha channel. 0 is the lowest quality, 100 is for lossless encoding. Default is 100/lossless.
- **SPEED=[0-10]:** Defaults to `6`. Speed of encoding. 0=slowest. 10=fastest.
- **NUM_THREADS=<integer>|ALL_CPUS:** Defaults to `ALL_CPUS`. Number of worker threads for compression.
- **SOURCE_ICC_PROFILE=value:** ICC profile encoded in Base64. Can also be set to empty string to avoid the ICC profile from the source dataset to be used.
- **WRITE_EXIF_METADATA=[YES/NO]:** Defaults to `YES`. Whether to write EXIF metadata present in source file.
- **WRITE_XMP=[YES/NO]:** Defaults to `YES`. Whether to write XMP metadata present in source file.
- **NBITS=[8/10/12/16]:** Bit depth. 16 only supported with libavif >= 1.4 and GDAL >= 3.13

- **YUV_SUBSAMPLING=[444/422/420]:** Defaults to 444. Type of *chroma subsampling <https://en.wikipedia.org/wiki/Chroma_subsampling>* to apply to YUV channels for RGB or RGBA images (it is ignored for single band of single band + alpha images) 4:4:4 corresponds to full horizontal and vertical resolution for chrominance channels. 4:2:2 corresponds to half horizontal and full vertical resolution. 4:2:0 corresponds to half horizontal and half vertical resolution. Only 4:4:4 can be used for lossless encoding.

6.6.5 See Also

- libavif

6.7 BAG -- Bathymetry Attributed Grid

Driver short name

BAG

Build dependencies

libhdf5

This driver provides read-only and creation support for bathymetry data in the BAG format. BAG files are actually a specific product profile in an HDF5 file, but a custom driver exists to present the data in a more convenient manner than is available through the generic HDF5 driver.

BAG files have two image bands representing Elevation (band 1), Uncertainty (band 2) values for each cell in a raster grid area. Nominal Elevation values may also be present, and starting with GDAL 3.2, any 2D array of numeric data type and with the same dimensions as Elevation present under BAG_root will be reported as a GDAL band.

The geotransform and coordinate system is extracted from the internal XML metadata provided with the dataset. However, some products may have unsupported coordinate system formats, if using the non-WKT way of encoding the spatial reference system.

The full XML metadata is available in the "xml:BAG" metadata domain.

Nodata, minimum and maximum values for each band are also reported.

6.7.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Added in version 3.2.

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.7.2 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). For open options specific to variable resolution, see following chapter.

Other open options are:

- **REPORT_VERTCRS=[YES/NO]:** (GDAL >= 3.2) Defaults to `YES`. To report the vertical CRS from BAG XML metadata as the vertical component of a compound CRS. If set to NO, only the horizontal part will be reported.

6.7.3 Variable resolution (VR) grid support

GDAL can handle BAG files with variable resolution grids. Such datasets are made of a low-resolution grid, which is the one presented by default by the driver, and for each of those low-resolution cells, a higher resolution grid can be present in the file. Such higher resolution grids are dubbed "supergrids" in GDAL.

The driver has different modes of working which can be controlled by the MODE open option:

MODE=[LOW_RES_GRID/LIST_SUPERGRIDS/RESAMPLED_GRID/INTERPOLATED/AUTO]: Defaults to `AUTO`. Driver mode.

- MODE=LOW_RES_GRID: this is the default mode (unless opening a supergrid). The driver will expose the low resolution grid, and indicate in the dataset metadata if the dataset has supergrids (HAS_SUPERGRIDS=TRUE), as well as the minimum and maximum resolution of those grids.
- MODE=LIST_SUPERGRIDS: in this mode, the driver will report the various supergrids in the subdataset list. It is possible to apply in this mode additional open options to restrict the search
  - **SUPERGRIDS_INDICES=<(y1,x1),(y2,x2),...>:** Tuple or list of tuples, of supergrids described by their y,x indices (starting from 0, y from the south of the grid, x from the west o the grid).
  - MINX=value: Minimum georeferenced X value to use as a filter for the supergrids to list.
  - MINY=value: Minimum georeferenced Y value to use as a filter for the supergrids to list.
  - MAXX=value: Maximum georeferenced X value to use as a filter for the supergrids to list.
  - MAXY=value: Maximum georeferenced Y value to use as a filter for the supergrids to list.
  - RES_FILTER_MIN=value: Minimum resolution of supergrids to take into account (excluded bound)
  - RES_FILTER_MAX=value: Maximum resolution of supergrids to take into account (included bound)
- Opening a supergrid. This mode is triggered by using as a dataset name a string formatted like BAG:my.bag:supergrid:{y}:{x}, which is the value of the SUBDATASET_x_NAME metadata items reported by the above described mode. {y} is the index (starting from 0, from the south of the grid), and {x} is the index (starting from 0, from the west of the grid) of the supergrid to open.

- MODE=RESAMPLED_GRID: in this mode, the user specify the extent and resolution of a target grid, and for each cell of this target grid, the driver will find the nodes of the supergrids that fall into that cell. By default, it will select the node with the maximum elevation value to populate the cell value. Or if no node of any supergrid are found, the cell value will be set to the nodata value. Overviews are reported: note that, those overviews correspond to resampled grids computed with different values of the RESX and RESY parameters, but using the same value population rules (and not nearest neighbour resampling of the full resolution resampled grid). The last overview level is the low-resolution grid (if SUPERGRIDS_MASK=NO)

  The available open options in this mode are:

  - MINX=value: Minimum georeferenced X value for the resampled grid. By default, the corresponding value of the low resolution grid.
  - MINY=value: Minimum georeferenced Y value for the resampled grid. By default, the corresponding value of the low resolution grid.
  - MAXX=value: Maximum georeferenced X value for the resampled grid. By default, the corresponding value of the low resolution grid.
  - MAXY=value: Maximum georeferenced Y value for the resampled grid. By default, the corresponding value of the low resolution grid.
  - RESX=value: Horizontal resolution. By default, and if RES_STRATEGY is set to AUTO, this will be the minimum resolution among all the supergrids.
  - RESY=value: Vertical resolution (positive value). By default, and if RES_STRATEGY is set to AUTO, this will be the minimum resolution among all the supergrids.
  - RES_STRATEGY=AUTO/MIN/MAX/MEAN: Which strategy to apply to set the resampled grid resolution. By default, if none of RESX, RESY, RES_FILTER_MIN and RES_FILTER_MAX is specified, the AUTO strategy will correspond to the MIN strategy: that is the minimum resolution among all the supergrids is used. If MAX is specified, the maximum resolution among all the supergrids is used. If MEAN is specified, the mean resolution among all the supergrids is used. RESX and RESY, if defined, will override the resolution determined by RES_STRATEGY.
  - RES_FILTER_MIN=value: Minimum resolution of supergrids to take into account (excluded bound, except if it is the minimum resolution of supergrids). By default, the minimum resolution of supergrids available. If this value is specified and none of RES_STRATEGY, RES_FILTER_MAX, RESX or RESY is specified, the maximum resolution among all the supergrids will be used as the resolution for the resampled grid.
  - RES_FILTER_MAX=value: Maximum resolution of supergrids to take into account (included bound). By default, the maximum resolution of supergrids available. If this value is specified and none of RES_STRATEGY, RESX or RESY is specified, this will also be used as the resolution for the resampled grid.
  - VALUE_POPULATION=MIN/MAX/MEAN/COUNT: Which value population strategy to apply to compute the resampled cell values. This default to MAX: the elevation value of a target cell is the maximum elevation of all supergrid nodes (potentially filtered with RES_FILTER_MIN and/or RES_FILTER_MAX) that fall into this cell; the corresponding uncertainty will be the uncertainty of the source node where this maximum elevation si reached. If no supergrid node fall into the target cell, the nodata value is set. The MIN strategy is similar, except that this is the minimum elevation value among intersecting nodes that is selected. The MEAN strategy uses the mean value of the elevation of intersecting nodes, and the maximum uncertainty of those nodes. The COUNT strategy (GDAL >= 3.2) exposes one single UInt32 band where each target cell contains the count of supergrid nodes that fall into it.
  - SUPERGRIDS_MASK=YES/NO. Default to NO. If set to YES, instead of the elevation and uncertainty band, the dataset contains a single Byte band which is boolean valued. For a target cell, if at least one supergrids nodes (potentially filtered with RES_FILTER_MIN and/or RES_FILTER_MAX) falls into the cell, the cell value is set at 255. Otherwise it is set at 0. This can be used to distinguish if elevation

values at nodata are due to no source supergrid node falling into them, or if that/those supergrid nodes were themselves at the nodata value.

  - NODATA_VALUE=value. Override the default value, which is usually 1000000.

- MODE=INTERPOLATED: (GDAL >= 3.8) in this mode, the user specify the extent and resolution of a target grid, and for each cell of this target grid, the driver will interpolate the value from the surrounding nodes of the supergrid(s) that applies to the cell, using in priority bilinear interpolation (for a node that falls within a supergrid and when all 4 surrounding nodes of the supergrid are not at nodata value), and fallbacks to barycentric interpolation or inverse distance weighting in other situations.

  Overviews are reported. Note that those overviews correspond to resampled grids computed with different values of the RESX and RESY parameters, but for performance reasons, the interpolation is still limited to the immediate neighbours of the target grid in the supergrid(s), which result in the equivalent of nearest-neighbour downsampling of the full resolution raster. The last overview level is the low-resolution grid.

  The available open options in this mode are MINX, MINY, MAXX, MAXY, RESX, RESY, RES_STRATEGY, RES_FILTER_MIN, RES_FILTER_MAX and NODATA_VALUE (cf their above description for the MODE=RESAMPLED_GRID)

6.7.4 Spatial metadata support

Starting with GDAL 3.2, GDAL can expose BAG files with spatial metadata.

When such spatial metadata is present, the subdataset list will include names of the form 'BAG:"{filename}":georef_metadata:{name_of_layer}' where `name_of_layer` is the name of a subgroup under `/BAG_root/Georef_metadata`

The values of the `keys` dataset under each metadata layer are used as the GDAL raster value. And the corresponding `values` dataset is exposed as a GDAL Raster Attribute Table associated to the GDAL raster band. If `keys` is absent, record 1 of `values` is assumed to be met for each elevation point that does not match the nodata value of the elevation band.

When variable resolution grids are present, the MODE=LIST_SUPERGRIDS open option will cause subdatasets of names of the form 'BAG:"{filename}":georef_metadata:{name_of_layer}:{y}:{x}' to be reported. When opening such a subdataset, the `varres_keys` dataset will be used to populate the GDAL raster value. If `varres_keys` is absent, record 1 of `values` is assumed to be met for each elevation point that does not match the nodata value of the variable resolution elevation band.

6.7.5 Tracking list support

When the dataset is opened in vector mode (ogrinfo, ogr2ogr, etc.), the tracking_list dataset will be reported as a OGR vector layer

6.7.6 Creation support

The driver can create a BAG dataset (without variable resolution extension) with the elevation and uncertainty bands from a source dataset. The source dataset must be georeferenced, and have one or two bands. The first band is assumed to be the elevation band, and the second band the uncertainty band. If the second band is missing, the uncertainty will be set to nodata.

The driver will instantiate the BAG XML metadata by using a template file, which is by default, bag_template.xml, found in the GDAL data definition files. This template contains variables, present as ${KEYNAME} or ${KEYNAME:default_value} in the XML file, that can be substituted by providing a creation option whose name is the VAR_ string prefixed to the key name.

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following creation options are supported:

- **VAR_INDIVIDUAL_NAME=value:** Defaults to `unknown`. String to fill contact/CI_ResponsibleParty/individualName.
- **VAR_ORGANISATION_NAME=value:** Defaults to `unknown`. String to fill contact/CI_ResponsibleParty/organisationName.
- **VAR_POSITION_NAME=value:** Defaults to `unknown`. String to fill contact/CI_ResponsibleParty/positionName.
- **VAR_DATE=<YYYY-MM-DD>:** Defaults to `current` date. Value to fill dateStamp/Date.
- **VAR_VERT_WKT=value:** WKT string to fill referenceSystemInfo/MD_ReferenceSystem/referenceSystemIdentifier/RS_Identifi for the vertical coordinate reference system. If not provided, and if the input CRS is not a compound CRS, default to VERT_CS["unknown", VERT_DATUM["unknown", 2000]].
- **VAR_ABSTRACT=value:** Defaults to `<empty` string>. String to fill identificationInfo/abstract.
- **VAR_PROCESS_STEP_DESCRIPTION=value:** Defaults to `Generated` by GDAL x.y.z. String to fill dataQualityInfo/lineage/LI_Lineage/processStep/LI_ProcessStep/description.
- **VAR_DATETIME=<YYYY-MM-DDTHH:MM:SS>:** Defaults to `current` datetime. Value to fill dataQualityInfo/lineage/LI_Lineage/processStep/LI_ProcessStep/dateTime/DateTime.
- **VAR_RESTRICTION_CODE=<enumerated_value>:** Defaults to `otherRestrictions`. Value to fill metadataConstraints/MD_LegalConstraints/useConstraints/MD_RestrictionCode.
- **VAR_OTHER_CONSTRAINTS=value:** Defaults to `unknown`. String to fill metadataConstraints/MD_LegalConstraints/otherConstraints.
- **VAR_CLASSIFICATION=<enumerated_value>:** Defaults to `unclassified`. Value to fill metadataConstraints/MD_SecurityConstraints/classification/MD_ClassificationCode.
- **VAR_SECURITY_USER_NOTE=value:** Defaults to `none`. String to fill metadataConstraints/MD_SecurityConstraints/userNote.

Other required variables found in the template, such as RES, RESX, RESY, RES_UNIT, HEIGHT, WIDTH, CORNER_POINTS and HORIZ_WKT will be automatically filled from the input dataset metadata.

The other following creation options are available:

- **TEMPLATE=<filename>:** Path to a XML file that can serve as a template. This will typically be a customized version of the base bag_template.xml file. The file can contain other substitutable variables than the ones mentioned above by using a similar syntax.
- **VAR_xxxx=value:** Substitute variable `${xxxx}` in the template XML value by the provided value.
- **BAG_VERSION=value:** Defaults to `1.6.2`. Value to write in the /BAG_root/BAG Version attribute.
- **COMPRESS=[NONE/DEFLATE]:** Defaults to `DEFLATE`. Compression for elevation and uncertainty grids.
- **ZLEVEL=1-9:** Defaults to `6`. Deflate compression level.
- **BLOCK_SIZE=<integer>:** Chunking size of the HDF5 arrays. Default to 100, or the maximum dimension of the raster if smaller than 100.

6.7.7 Usage examples

- Opening in low resolution mode:

```
$ gdalinfo data/test_vr.bag

[...]
Size is 6, 4
```

(continues on next page)

(continued from previous page)

```
[...]
  HAS_SUPERGRIDS=TRUE
  MAX_RESOLUTION_X=29.900000
  MAX_RESOLUTION_Y=31.900000
  MIN_RESOLUTION_X=4.983333
  MIN_RESOLUTION_Y=5.316667
[...]
```

- Displaying available supergrids:

```
$ gdalinfo data/test_vr.bag -oo MODE=LIST_SUPERGRIDS

[...]
Subdatasets:
  SUBDATASET_1_NAME=BAG:"data/test_vr.bag":supergrid:0:0
  SUBDATASET_1_DESC=Supergrid (y=0, x=0) from (x=70.100000,y=499968.100000) to␣
↪(x=129.900000,y=500031.900000), resolution (x=29.900000,y=31.900000)
  SUBDATASET_2_NAME=BAG:"data/test_vr.bag":supergrid:0:1
  SUBDATASET_2_DESC=Supergrid (y=0, x=1) from (x=107.575000,y=499976.075000) to␣
↪(x=152.424999,y=500023.924999), resolution (x=14.950000,y=15.950000)
[...]
  SUBDATASET_24_NAME=BAG:"data/test_vr.bag":supergrid:3:5
  SUBDATASET_24_DESC=Supergrid (y=3, x=5) from (x=232.558335,y=500077.391667) to␣
↪(x=267.441666,y=500114.608334), resolution (x=4.983333,y=5.316667)
[...]
```

- Opening a particular supergrid:

```
$ gdalinfo BAG:"data/test_vr.bag":supergrid:3:5
```

- Converting a BAG in resampling mode with default parameters (use of minimum resolution of supergrids, MAX value population rule):

```
$ gdal_translate data/test_vr.bag -oo MODE=RESAMPLED_GRID out.tif
```

- Converting a BAG in resampling mode with a particular grid origin and resolution

```
$ gdal_translate data/test_vr.bag -oo MODE=RESAMPLED_GRID -oo MINX=80 -oo␣
↪MINY=500000 -oo RESX=16 -oo RESY=16 out.tif
```

- Converting a BAG in resampling mode, with a mask indicating where supergrids nodes intersect the cell of the resampled dataset.

```
$ gdal_translate data/test_vr.bag -oo MODE=RESAMPLED_GRID -oo SUPERGRIDS_MASK=YES␣
↪out.tif
```

- Converting a BAG in resampling mode, by filtering on supergrid resolutions (and the resampled grid will use 4 meter resolution by default)

```
$ gdal_translate data/test_vr.bag -oo MODE=RESAMPLED_GRID -oo RES_FILTER_MIN=4 -oo␣
↪RES_FILTER_MAX=8 out.tif
```

- Converting a GeoTIFF file to a BAG dataset, and provide a custom value for the ABSTRACT substitutable variable.

```
$ gdal_translate in.tif out.bag -co "VAR_ABSTRACT=My abstract"
```

- Converting a (VR) BAG in resampling mode with a particular grid resolution (5m) to a BAG dataset (without variable resolution extension), and provide a custom value for the ABSTRACT metadata:

```
$ gdal_translate data/test_vr.bag -oo MODE=RESAMPLED_GRID -oo RESX=5 -oo RESY=5 out.
↪bag -co "VAR_ABSTRACT=My abstract"
```

- Displaying the tracking list:

```
$ ogrinfo -al data/my.bal
```

6.7.8 See Also

- Implemented as bagdataset.cpp.
- The Open Navigation Surface Project
- Description of Bathymetric Attributed Grid Object (BAG) Version 1.6
- Variable resolution grid extension for BAG files
- *S-102 driver*

6.8 BASISU -- Basis Universal

Added in version 3.6.

Driver short name

BASISU

Build dependencies

Basis Universal

From https://github.com/BinomialLLC/basis_universal,

> Basis Universal is a "supercompressed" GPU texture data interchange system that supports two highly compressed intermediate file formats (.basis or the .KTX2 open standard from the Khronos Group) that can be quickly transcoded to a very wide variety of GPU compressed and uncompressed pixel formats.

This driver handles textures with the .basis extension. For KTX2, refer to the *KTX2* driver.

Note that while the file format supports direct transcoding to other GPU pixel formats, this GDAL driver supports only conversion between uncompressed RGB(A) data and Basis Universal textures.

When a file is made of several images, they are exposed as subdataset, with the *BASISU:filename:image_idx* syntax.

Mipmaps levels are exposed as GDAL overviews.

6.8.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.8.2 Creation options

The format supports 2 modes: a high quality mode which is internally based off the UASTC compressed texture format, and the original lower quality mode which is based off a subset of ETC1 called "ETC1S". The default is ETC1S. Only input of type Byte and with 1 (grey), 2 (grey+alpha), 3 (RGB) or 4 (RGB + alpha) bands is supported. Refer to https://github.com/BinomialLLC/basis_universal for more details on those modes and their options.

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following creation options are available:

- **COMPRESSION=[ETC1S/UASTC]:** Defaults to `ETC1S`.
- **UASTC_LEVEL=[0/1/2/3/4]:** Defaults to `2`. The higher value, the higher the quality but the slower computing time. 4 is impractically slow. Only valid when *COMPRESSION=UASTC*.
- **UASTC_RDO_LEVEL=<float>:** Defaults to `1.0`. Rate distortion optimization level. The lower value, the higher the quality, but the larger the file size. Usual range is [0.2,3]. Only valid when *COMPRESSION=UASTC*.
- **ETC1S_LEVEL=[0/1/2/3/4/5/6]:** Defaults to `1`. The higher value, the higher the quality but the slower computing time. Only valid when *COMPRESSION=ETC1S*.
- **ETC1S_QUALITY_LEVEL=[<integer in [1/255] range>]:** Defaults to `128`. The higher value, the higher the quality, but the larger the file size. Only valid when *COMPRESSION=ETC1S*.
- **ETC1S_MAX_ENDPOINTS_CLUSTERS=[<integer in [1/16128] range>]:** Maximum number of endpoint clusters. When set, *ETC1S_MAX_SELECTOR_CLUSTERS* must also be set. Mutually exclusive with *ETC1S_QUALITY_LEVEL*. Only valid when *COMPRESSION=ETC1S*.
- **ETC1S_MAX_SELECTOR_CLUSTERS=[<integer in [1/16128] range>]:** Maximum number of selector clusters. When set, *ETC1S_MAX_ENDPOINTS_CLUSTERS* must also be set. Mutually exclusive with *ETC1S_QUALITY_LEVEL*. Only valid when *COMPRESSION=ETC1S*.
- **NUM_THREADS=<integer>:** Defaults to the maximum number of virtual CPUs available. Can also be controlled with the *GDAL_NUM_THREADS* configuration option.
- **MIPMAP=[YES/NO]:** Defaults to `NO`. Whether to enable MIPMAP generation.
- **COLORSPACE=[PERCEPTUAL_SRGB/LINEAR]:** Defaults to `PERCEPTUAL_SRGB`. For non-photometric input, use LINEAR to avoid unnecessary artifacts.

6.8.3 Build instructions

Building basisu as a library requires currently building the *cmake* branch of the https://github.com/rouault/basis_universal/tree/cmake fork.

```
git clone -b cmake https://github.com/rouault/basis_universal
cd basis_universal
mkdir build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=/path/to/install-basisu -DCMAKE_BUILD_TYPE=Release -
↪DBUILD_SHARED_LIBS=ON
cmake --build . --config Release --target install
```

Once basisu is built, GDAL CMake options must be configured by pointing the basisu install prefix in the `CMAKE_PREFIX_PATH` variable or `basisu_ROOT` variable.

6.9 BMP -- Microsoft Windows Device Independent Bitmap

Driver short name

BMP

Driver built-in by default

This driver is built-in by default

MS Windows Device Independent Bitmaps supported by the Windows kernel and mostly used for storing system decoration images. Due to the nature of the BMP format it has several restrictions and could not be used for general image storing. In particular, you can create only 1-bit monochrome, 8-bit pseudocoloured and 24-bit RGB images only. Even grayscale images must be saved in pseudocolour form.

This driver supports reading almost any type of the BMP files and could write ones which should be supported on any Windows system. Only single- or three- band files could be saved in BMP file. Input values will be resampled to 8 bit.

If an ESRI world file exists with the .bpw, .bmpw or .wld extension, it will be read and used to establish the geotransform for the image.

6.9.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.9.2 Creation Options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following creation options are supported:

- **WORLDFILE=YES:** Force the generation of an associated ESRI world file (with the extension .wld).

6.9.3 See Also

- Implemented as bmpdataset.cpp.
- Wikipedia BMP file format

6.10 BSB -- Maptech/NOAA BSB Nautical Chart Format

Driver short name

BSB

Driver built-in by default

This driver is built-in by default

BSB Nautical Chart format is supported for read access, including reading the colour table and the reference points (as GCPs). Note that the .BSB files cannot be selected directly. Instead select the .KAP files. Versions 1.1, 2.0 and 3.0 have been tested successfully.

This driver should also support GEO/NOS format as supplied by Softchart. These files normally have the extension .nos with associated .geo files containing georeferencing ... the .geo files are currently ignored.

This driver is based on work by Mike Higgins. See the bsb_read.c files for details on patents affecting BSB format.

It is possible to select an alternate color palette via the BSB_PALETTE configuration option. The default value is RGB. Other common values that can be found are : DAY, DSK, NGT, NGR, GRY, PRC, PRG...

NOTE: Implemented as bsbdataset.cpp.

6.10.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.10.2 Metadata

The following metadata items may be reported:

- **BSB_KNP**: content of the KNP/ header field, giving information on the coordinate reference system.
- **BSB_KNQ**: content of the KNQ/ header field, giving information on the coordinate reference system.
- **BSB_CUTLINE**: (starting with GDAL 3.1). When PLY/ header is present, Well-Known text representation of a polygon with coordinates in longitude, latitude order, representing the cutline of the chart.

6.11 BT -- VTP .bt Binary Terrain Format

Driver short name

BT

Driver built-in by default

This driver is built-in by default

The .bt format is used for elevation data in the VTP software. The driver includes support for reading and writing .bt 1.3 format including support for Int16, Int32 and Float32 pixel data types.

The driver does **not** support reading or writing gzipped (.bt.gz) .bt files even though this is supported by the VTP software. Please unpack the files before using with GDAL using the "gzip -d file.bt.gz".

Projections in external .prj files are read and written, and support for most internally defined coordinate systems is also available.

Read/write imagery access with the GDAL .bt driver is terribly slow due to a very inefficient access strategy to this column oriented data. This could be corrected, but it would be a fair effort.

NOTE: Implemented as btdataset.cpp.

See Also: The BT file format is defined on the VTP web site.

6.11.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.12 BYN - Natural Resources Canada's Geoid file format (.byn)

Driver short name

BYN

Driver built-in by default

This driver is built-in by default

Files with extension ".byn" have a binary format. The format includes two sections which are the Header and the Data. The data are stored by rows starting from the north. Each row is stored from the west to the east. The data are either short (2 bytes) or standard (4 bytes) integers. The size of the bytes is defined in the header.

The total size of the file is 80 bytes + (Row x Column x (2 or 4) bytes) where Row is the number of rows in the grid and Column is the number of columns in the grid. Row and Column can be calculated by these two equations:

Row = (North Boundary - South Boundary) / (NS Spacing) + 1

Column = (East Boundary - West Boundary) / (EW Spacing) + 1

The ".byn" files may contain undefined data. Depending if the data are stored as 2-byte or 4-byte integers, the undefined data are expressed the following way:

4-byte data (Standard integer): 9999.0 * Factor, the Factor is given in the header

2 byte data (Short integer): 32767

Most of the parameters in the ".byn" header can be read by clicking the “Information” icon in software GPS-H.

NOTE: Files with extension ".err" are also in the ".byn" format. An ".err" file usually contains the error estimates of the ".byn" file of the same name (e.g., CGG2013n83.byn and CGG2013n83.err). The ".err" file will have variable Data equal to 1 or 3.

6.12.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.12.2 Factor

When translating from BYN file to or from another formats the scale will affect the result profoundly.

Translating to a format that supports Scale (GTIFF for example) will maintain the data type and thescale information. The pixel values are perfectly preserved.

Translating to a format that does not support Scale will maintain the data type but without the Scale, meaning loss of information on every pixels.

The solution to the problem above is to use "-unscale" and "-ot Float32" when using gdal_translate or GDAL API. That will produce a dataset without scale but with the correct pixel information. Ex.:

gdal_translate CGG2013an83.err test2.tif -unscale -ot Float32

NOTE: The BYN header variable **Factor** is the inverse of GDAL **Scale**. (Scale = 1.0 / Factor).

6.12.3 See Also

- Implemented as byndataset.cpp.
- www.nrcan.gc.ca

6.13 CAD -- AutoCAD DWG raster layer

Driver short name

CAD

Build dependencies

(internal libopencad provided)

OGR DWG support is based on libopencad, so the list of supported DWG (DXF) versions can be seen in libopencad documentation. All drawing entities are separated into layers as they are in DWG file. The rasters are usually a separate georeferenced files (GeoTiff, Jpeg, Png etc.) which exist in DWG file as separate layers. The driver try to get spatial reference and other methadata from DWG Image description and set it to GDALDataset.

NOTE: Implemented as gdalcaddataset.cpp.

6.13.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.14 CALS -- CALS Type 1

Driver short name

CALS

Driver built-in by default

This driver is built-in by default

CALS Type 1 rasters are untiled black and white rasters made of a 2048-byte text header in the MIL-STD-1840 standard, followed by a single datastream compressed with CCITT/ITU-T Recommendation 6, aka Group 6, aka CCITT FAX 4. CALS Type 1 rasters are one of the 4 types of formats described by MIL-PRF-28002C (this standard is now deprecated). Other types are not handled by this driver.

This driver supports reading and creation of CALS Type 1 rasters. Update of existing rasters is not supported.

A CALS dataset is exposed by the driver as a single-band 1-bit raster with a 2-entry color table. The first entry (0) is white (RGB=255,255,255) and the second entry (1) is black (RGB=0,0,0).

6.14.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.14.2 Metadata

The following metadata items might be exposed by the driver (or read from the input dataset to generate a cor

- **PIXEL_PATH**: First value of the "rorient" header field, measured in degrees counterclockwise from the positive horizontal axis (east). The pel path value represents the number of degrees the image would have to be rotated counterclockwise in order to display the image with proper viewing orientation. The permissible values for the pixel path direction shall be "0", "90", "180", or "270". 0 is the typical value. If PIXEL_PATH=0 and LINE_PROGRESSION=270, neither are reported.
- **LINE_PROGRESSION**: Second value of the "rorient" header field. The line progression direction is measured in degrees counterclockwise from the pel path direction. The permissible values for the line progression direction shall be "90" or "270". 270 is the typical value. If PIXEL_PATH=0 and LINE_PROGRESSION=270, neither are reported.

- **TIFFTAG_XRESOLUTION**: Scan X resolution in dot per inch, from the "rdensty" header field. TIFFTAG_XRESOLUTION and TIFFTAG_YRESOLUTION are always equal in CALS.
- **TIFFTAG_YRESOLUTION**: Scan Y resolution in dot per inch, from the "rdensty" header field. TIFFTAG_XRESOLUTION and TIFFTAG_YRESOLUTION are always equal in CALS.

6.14.3 Creation issues

Only single band 1-bit rasters are valid input to create a new CALS file. If the input raster has no color table, 0 is assumed to be black and 1 to be white. If the input raster has a (2 entries) color table, the value for the black and white color will be determined from the color table.

6.14.4 See Also

- MIL-PRF-28002C
- MIL-STD-1840C

6.15 CEOS -- CEOS Image

Driver short name

CEOS

Driver built-in by default

This driver is built-in by default

This is a simple, read-only reader for ceos image files. To use, select the main imagery file. This driver reads only the image data, and does not capture any metadata, or georeferencing.

This driver is known to work with CEOS data produced by Spot Image, but will have problems with many other data sources. In particular, it will only work with eight bit unsigned data.

See the separate *SAR_CEOS -- CEOS SAR Image* driver for access to SAR CEOS data products.

NOTE: Implemented as ceosdataset.cpp.

6.15.1 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.16 COASP -- DRDC COASP SAR Processor Raster

Driver short name

COASP

Driver built-in by default

This driver is built-in by default

NOTE: Implemented as coasp_dataset.cpp.

6.17 COG -- Cloud Optimized GeoTIFF generator

Added in version 3.1.

Driver short name

COG

Driver built-in by default

This driver is built-in by default

This driver supports the creation of Cloud Optimized GeoTIFF (COG)

It essentially relies upon the *GTiff -- GeoTIFF File Format* driver with the `COPY_SRC_OVERVIEWS=YES` creation option, but automatically does the needed preprocessing stages (reprojection if asked and creation of overviews on imagery and/or mask) if not already done, and also takes care of morphing the input dataset into the expected form when using some compression types (for example a RGBA dataset will be transparently converted to a RGB+mask dataset when selecting JPEG compression)

This driver is nominally used through the *`GDALDriver::CreateCopy()`* method, that natively does not support random writing but requires a source dataset to be provided as input. During the writing process, if overviews are generated, they will be created in a temporary dataset, requiring storage capacity of roughly one third of the final file size, before being transferred to the output file. By default temporary files are created in the same directory as the final file, if the target file system supports random writing and if the *`CPL_TMPDIR`* configuration option is not set.

Starting with GDAL 3.13, the *`GDALDriver::Create()`* method is also implemented, by using a temporary GeoTIFF dataset, which means that at least twice the final size of the file of storage capacity is required. Note that as the temporary dataset uses lossless compression, if the final COG file uses lossy compression, more temporary storage may be needed.

Validation whether a TIFF file is a valid COG file can be done with *gdal driver cog validate*.

6.17.1 Driver capabilities

Supports Create()

Added in version 3.13.

Supports CreateCopy()

This driver supports the *`GDALDriver::CreateCopy()`* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.17.2 Creation Options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python).

6.17.2.1 General creation options

- **BLOCKSIZE=<integer>:** Defaults to `512`. Sets the tile width and height in pixels. Must be divisible by 16.
- **INTERLEAVE=[BAND/PIXEL/TILE]:** (GDAL >= 3.11) Set the interleaving to use. See *Multiband pixel organization (INTERLEAVE metadata item)* for more details.
  - `PIXEL`: for each spatial block, one TIFF tile/strip gathering values for all bands is used . This matches the `contiguous` planar configuration in TIFF terminology. This is also known as a `BIP (Band Interleaved Per Pixel)` organization. Such method is the default, and may be slightly less efficient than BAND interleaving for some purposes, but some applications only support pixel interleaved TIFF files. On the other hand, image-based compression methods may perform better using PIXEL interleaving. For JPEG PHOTOMETRIC=YCbCr, pixel interleaving is required. It is also required for WebP compression.

    Prior to GDAL 3.11, INTERLEAVE=PIXEL was the only possible interleaving output by the COG driver.

    Assuming a pixel[band][y][x] indexed array, when using CreateCopy(), the pseudo code for the file disposition is:

```
for y in 0 ... numberOfBlocksInHeight - 1:
    for x in 0 ... numberOfTilesInWidth - 1:
        for j in 0 ... blockHeight - 1:
            for i in 0 ... blockWidth -1:
                start_new_strip_or_tile()
                for band in 0 ... numberBands -1:
                    write(pixel[band][y*blockHeight+j][x*blockWidth+i])
                end_new_strip_or_tile()
            end_for
        end_for
    end_for
end_for
```

  - `BAND`: for each spatial block, one TIFF tile/strip is used for each band. This matches the contiguous `separate` configuration in TIFF terminology. This is also known as a `BSQ (Band SeQuential)` organization.

    In addition to that, when using CreateCopy(), data for the first band is written first, followed by data for the second band, etc. The pseudo code for the file disposition is:

```
for y in 0 ... numberOfBlocksInHeight - 1:
    for x in 0 ... numberOfTilesInWidth - 1:
```

(continues on next page)

(continued from previous page)

```
        start_new_strip_or_tile()
        for band in 0 ... numberBands -1:
            for j in 0 ... blockHeight - 1:
                for i in 0 ... blockWidth -1:
                    write(pixel[band][y*blockHeight+j][x*blockWidth+i])
                end_for
            end_for
        end_new_strip_or_tile()
    end_for
end_for
```

- TILE (added in 3.11): this is a sort of compromise between PIXEL and BAND, using the separate configuration, but where data for a same spatial block is written for all bands, before the data of the next spatial block. When the block height is 1, this is also known as a BIL (Band Interleaved per Line) organization.

  Such a layout may be useful for writing hyperspectral datasets (several hundred of bands), to get a compromise between efficient spatial query, and partial band selection.

  Assuming a pixel[band][y][x] indexed array, when using CreateCopy(), the pseudo code for the file disposition is:

```
for y in 0 ... numberOfBlocksInHeight - 1:
    for x in 0 ... numberOfTilesInWidth - 1:
        for band in 0 ... numberBands -1:
            start_new_strip_or_tile()
            for j in 0 ... blockHeight - 1:
                for i in 0 ... blockWidth -1:
                    write(pixel[band][y*blockHeight+j][x*blockWidth+i])
                end_for
            end_for
            end_new_strip_or_tile()
        end_for
    end_for
end_for
```

Starting with GDAL 3.5, when copying from a source dataset with multiple bands which advertises a INTERLEAVE metadata item, if the INTERLEAVE creation option is not specified, the source dataset INTERLEAVE will be automatically taken into account, unless the COMPRESS creation option is specified.

- **COMPRESS=[NONE/LZW/JPEG/DEFLATE/ZSTD/WEBP/LERC/LERC_DEFLATE/LERC_ZSTD/LZMA]:** Defaults to LZW. Set the compression to use. Defaults to LZW starting with GDAL 3.4 (default in previous version is NONE).
  - JPEG should generally only be used with Byte data (8 bit per channel). But if GDAL is built with internal libtiff and libjpeg, it is possible to read and write TIFF files with 12bit JPEG compressed TIFF files (seen as UInt16 bands with NBITS=12). For the COG driver, JPEG compression for 3 or 4-band images automatically selects the PHOTOMETRIC=YCBCR colorspace with a 4:2:2 subsampling of the Y,Cb,Cr components with the default INTERLEAVE=PIXEL. For a input dataset (single-band or 3-band), plus an alpha band, the alpha band will be converted as a 1-bit DEFLATE compressed mask.
  - LZW, DEFLATE and ZSTD compressions can be used with the PREDICTOR creation option.
  - ZSTD is available when using internal libtiff and if GDAL built against libzstd >=1.0, or if built against external libtiff with zstd support.

  - `WEBP` is available when using internal libtiff and if GDAL built against libwebp, or if built against external libtiff with WebP support. It can only be used with the default INTERLEAVE=PIXEL.
  - `LERC` is available when using internal libtiff.
  - `LERC_ZSTD` is available when `LERC` and `ZSTD` are available.
  - `JXL` is for JPEG-XL, and is only available when using internal libtiff and building GDAL against https://github.com/libjxl/libjxl . JXL compression may only be used on datasets with 4 bands or less. Option added in GDAL 3.4

- **LEVEL=<integer>:** DEFLATE/ZSTD/LERC_DEFLATE/LERC_ZSTD/LZMA compression level. A lower number will result in faster compression but less efficient compression rate. 1 is the fastest.
  - For DEFLATE/LZMA, 9 is the slowest/higher compression rate (or 12 when using a libtiff with libdeflate support). The default is 6.
  - For ZSTD, 22 is the slowest/higher compression rate. The default is 9.
- **MAX_Z_ERROR=<threshold>:** Defaults to `0`. Set the maximum error threshold on values for LERC/LERC_DEFLATE/LERC_ZSTD compression. The default is 0 (lossless).
- **MAX_Z_ERROR_OVERVIEW=<threshold>:** (GDAL >= 3.8) Set the maximum error threshold on values for LERC/LERC_DEFLATE/LERC_ZSTD compression, on overviews. The default is the value of *MAX_Z_ERROR*
- **QUALITY=<integer>:** Defaults to `75`. JPEG/WEBP quality setting. A value of 100 is best quality (least compression), and 1 is worst quality (best compression). For WEBP, QUALITY=100 automatically turns on lossless mode.
- **JXL_LOSSLESS=[YES/NO]:** Defaults to `YES`. Set whether JPEG-XL compression should be lossless (YES) or lossy (NO). For lossy compression, the underlying data should be either gray, gray+alpha, rgb or rgb+alpha.
- **JXL_EFFORT=1-9:** Defaults to `5`. Level of effort for JPEG-XL compression. The higher, the smaller file and slower compression time.
- **JXL_DISTANCE=0.01-25:** Defaults to `1.0`. (Only applies when JXL_LOSSLESS=NO) Distance level for lossy JPEG-XL compression. It is specified in multiples of a just-noticeable difference (cf butteraugli for the definition of the distance) That is, 0 is mathematically lossless, 1 should be visually lossless, and higher distances yield denser and denser files with lower and lower fidelity. The recommended range is [0.5,3].
- **JXL_ALPHA_DISTANCE=[-1/0/0.01-25]:** (GDAL >= 3.7) Defaults to `-1`. (Only applies when JXL_LOSSLESS=NO) (libjxl > 0.8.1) Distance level for alpha channel for lossy JPEG-XL compression. It is specified in multiples of a just-noticeable difference. (cf butteraugli for the definition of the distance) That is, 0 is mathematically lossless, 1 should be visually lossless, and higher distances yield denser and denser files with lower and lower fidelity. For lossy compression, the recommended range is [0.5,3]. The default value is the special value -1.0, which means to use the same distance value as non-alpha channel (ie *JXL_DISTANCE*).
- **NUM_THREADS=[<number_of_threads>/ALL_CPUS]:** Enable multi-threaded compression by specifying the number of worker threads. Default is compression in the main thread. This also determines the number of threads used when reprojection is done with the *TILING_SCHEME* or *TARGET_SRS* creation options. (Overview generation is also multithreaded since GDAL 3.2)
- **NBITS=<integer>:** (GDAL >= 3.7) Create a file with less than 8 bits per sample by passing a value from 1 to 7. The apparent pixel type should be Byte. Values of n=9...15 (UInt16 type) and n=17...31 (UInt32 type) are also accepted. n=16 is accepted for Float32 type to generate half-precision floating point values.
- **PREDICTOR=[YES/NO/STANDARD/FLOATING_POINT]:** Defaults to `NO`. Set the predictor for LZW, DEFLATE and ZSTD compression. If YES is specified, then standard predictor (Predictor=2) is used for integer data type, and floating-point predictor (Predictor=3) for floating point data type (in some circumstances, the standard predictor might perform better than the floating-point one on floating-point data). STANDARD or FLOATING_POINT can also be used to select the precise algorithm wished.

- **BIGTIFF=[YES/NO/IF_NEEDED/IF_SAFER]:** Control whether the created file is a BigTIFF or a classic TIFF.
  - `YES` forces BigTIFF.
  - `NO` forces classic TIFF.
  - `IF_NEEDED` will only create a BigTIFF if it is clearly needed (in the uncompressed case, and image larger than 4GB. So no effect when using a compression).
  - `IF_SAFER` will create BigTIFF if the resulting file *might* exceed 4GB. Note: this is only a heuristics that might not always work depending on compression ratios.

  BigTIFF is a TIFF variant which can contain more than 4GiB of data (size of classic TIFF is limited by that value). This option is available if GDAL is built with libtiff library version 4.0 or higher. The default is IF_NEEDED.

  When creating a new GeoTIFF with no compression, GDAL computes in advance the size of the resulting file. If that computed file size is over 4GiB, GDAL will automatically decide to create a BigTIFF file. However, when compression is used, it is not possible in advance to known the final size of the file, so classical TIFF will be chosen. In that case, the user must explicitly require the creation of a BigTIFF with BIGTIFF=YES if the final file is anticipated to be too big for classical TIFF format. If BigTIFF creation is not explicitly asked or guessed and the resulting file is too big for classical TIFF, libtiff will fail with an error message like "TIFFAppendToStrip:Maximum TIFF file size exceeded".

- **RESAMPLING=[NEAREST/AVERAGE/BILINEAR/CUBIC/CUBICSPLINE/LANCZOS/MODE/RMS]:** Resampling method used for overview generation or reprojection. For paletted images, NEAREST is used by default, otherwise it is CUBIC.

- **OVERVIEW_RESAMPLING=[NEAREST/AVERAGE/BILINEAR/CUBIC/CUBICSPLINE/LANCZOS/MODE/RMS]:** (GDAL >= 3.2) Resampling method used for overview generation. For paletted images, NEAREST is used by default, otherwise it is CUBIC. This overrides, for overview generation, the value of *RESAMPLING* if it specified.

- **WARP_RESAMPLING=[NEAREST/AVERAGE/BILINEAR/CUBIC/CUBICSPLINE/LANCZOS/MODE/RMS/MIN/MAX/MED/Q1/Q3]:** (GDAL >= 3.2) Resampling method used for reprojection. For paletted images, NEAREST is used by default, otherwise it is CUBIC. This overrides, for reprojection, the value of *RESAMPLING* if it specified.

- **OVERVIEWS=[AUTO/IGNORE_EXISTING/FORCE_USE_EXISTING/NONE]:** Defaults to `AUTO`. Describe the behavior regarding overview generation and use of source overviews.
  - `AUTO` (default): source overviews will be used if present. If not present, overviews will be automatically generated in the output file.
  - `IGNORE_EXISTING`: potential existing overviews on the source dataset will be ignored and new overviews will be automatically generated.
  - `FORCE_USE_EXISTING`: potential existing overviews on the source will be used. If there is no source overview, this is equivalent to specifying `NONE`.
  - `NONE`: potential source overviews will be ignored, and no overview will be generated.

  > **Note**
  >
  > When using the gdal_translate utility, source overviews will not be available if general options (i.e. options which are not creation options, like subsetting, etc.) are used.

- **OVERVIEW_COUNT=<integer>:** (GDAL >= 3.6) Number of overview levels to generate. This can be used to increase or decrease the number of levels in the COG file (when GDAL computes overviews from the full resolution dataset, that is when there are no source overviews or the user specifies *OVERVIEWS=IGNORE_EXISTING*),

or decrease the number of levels copied from the source dataset (in *OVERVIEWS=AUTO* or FORCE_USE_EXISTING modes when there are such overviews in the source dataset). If not specified, the driver will use all the overviews available in the source raster, in *OVERVIEWS=AUTO* or FORCE_USE_EXISTING modes. In situations where GDAL generates overviews, the default number of overview levels is such that the dimensions of the smallest overview are smaller or equal to the *BLOCKSIZE* value.

- **OVERVIEW_COMPRESS=[AUTO/NONE/LZW/JPEG/DEFLATE/ZSTD/WEBP/LERC/LERC_DEFLATE/LERC_ZSTD/LZMA]:** Defaults to AUTO. Set the compression method (see COMPRESS) to use when storing the overviews in the COG. By default (AUTO) the overviews will be created with the same compression method as the COG.
- **OVERVIEW_QUALITY=<integer>:** JPEG/WEBP quality setting. A value of 100 is best quality (least compression), and 1 is worst quality (best compression). By default the overviews will be created with the same quality as the COG, unless the compression type is different then the default is 75.
- **OVERVIEW_PREDICTOR=[YES/NO/STANDARD/FLOATING_POINT]:** Set the predictor for LZW, DEFLATE and ZSTD overview compression. By default the overviews will be created with the same predictor as the COG, unless the compression type of the overview is different, then the default is NO.
- **GEOTIFF_VERSION=[AUTO/1.0/1.1]:** Defaults to AUTO. Select the version of the GeoTIFF standard used to encode georeferencing information. 1.0 corresponds to the original 1995, GeoTIFF Revision 1.0, by Ritter & Ruth. 1.1 corresponds to the OGC standard 19-008, which is an evolution of 1.0, which clear ambiguities and fix inconsistencies mostly in the processing of the vertical part of a CRS. AUTO mode (default value) will generally select 1.0, unless the CRS to encode has a vertical component or is a 3D CRS, in which case 1.1 is used.

> **Note**
>
> Write support for GeoTIFF 1.1 requires libgeotiff 1.6.0 or later.

- **SPARSE_OK=[TRUE/FALSE]:** (GDAL >= 3.2) Defaults to FALSE. Should empty blocks be omitted on disk? When this option is set, any attempt of writing a block whose all pixels are 0 or the nodata value will cause it not to be written at all (unless there is a corresponding block already allocated in the file). Sparse files have 0 tile/strip offsets for blocks never written and save space; however, most non-GDAL packages cannot read such files. On the reading side, the presence of a omitted tile after a non-empty one may cause optimized readers to have to issue an extra GET request to the TileByteCounts array.
- **STATISTICS=[AUTO/YES/NO]:** (GDAL >= 3.8) Defaults to AUTO. Whether band statistics should be included in the output file. In AUTO mode, they will be included only if available in the source dataset. If setting to YES, they will always be included. If setting to NO, they will be never included.

6.17.2.2 Reprojection related creation options

- **TILING_SCHEME=[CUSTOM/GoogleMapsCompatible/...]:** Defaults to CUSTOM. If set to a value different than CUSTOM, the definition of the specified tiling scheme will be used to reproject the dataset to its CRS, select the resolution corresponding to the closest zoom level and align on tile boundaries at this resolution (the actual resolution can be controlled with the *ZOOM_LEVEL* or *ZOOM_LEVEL_STRATEGY* options). The tile size indicated in the tiling scheme definition (generally 256 pixels) will be used, unless the user has specified a value with the *BLOCKSIZE* creation option, in which case the user specified one will be taken into account (that is if setting a higher value than 256, the original tiling scheme is modified to take into account the size of the HiDPi tiles).

  In non-CUSTOM mode, TARGET_SRS, RES and EXTENT options are ignored. Starting with GDAL 3.2, the value of *TILING_SCHEME* can also be the filename of a JSON file according to the *OGC Two Dimensional Tile Matrix Set standard*, a URL to such file, the radical of a definition file in the GDAL data directory (e.g. FOO for a file named tms_FOO.json) or the inline JSON definition. The list of available tiling schemes can be obtained by looking at values of the TILING_SCHEME option reported by gdalinfo --format COG.

- **ZOOM_LEVEL=<integer>:** (GDAL >= 3.5) Zoom level number (starting at 0 for coarsest zoom level). Only used for *TILING_SCHEME* different from CUSTOM. If this option is specified, *ZOOM_LEVEL_STRATEGY* is ignored.
- **ZOOM_LEVEL_STRATEGY=[AUTO/LOWER/UPPER]:** (GDAL >= 3.2) Defaults to `AUTO`. Strategy to determine zoom level. Only used for *TILING_SCHEME* different from CUSTOM. LOWER will select the zoom level immediately below the theoretical computed non-integral zoom level, leading to subsampling. On the contrary, UPPER will select the immediately above zoom level, leading to oversampling. Defaults to AUTO which selects the closest zoom level.
- **TARGET_SRS=value:** to force reprojection of the input dataset to another SRS. The string can be a WKT string, a EPSG:XXXX code or a PROJ string.
- **RES=value:** Set the resolution of the target raster, in the units of *TARGET_SRS*. Only taken into account if *TARGET_SRS* is specified.
- **EXTENT=<minx,miny,maxx,maxy>:** Set the extent of the target raster, in the units of *TARGET_SRS*. Only taken into account if *TARGET_SRS* is specified.
- **ALIGNED_LEVELS=<integer>:** Number of resolution levels for which GeoTIFF tile and tiles defined in the tiling scheme match. When specifying this option, padding tiles will be added to the left and top sides of the target raster, when needed, so that a GeoTIFF tile matches with a tile of the tiling scheme. Only taken into account if *TILING_SCHEME* is different from CUSTOM. Effect of this option is only visible when setting it at 2 or more, since the full resolution level is by default aligned with the tiling scheme. For a tiling scheme whose consecutive zoom level resolutions differ by a factor of 2, care must be taken in setting this value to a high number of levels, as up to 2^(ALIGNED_LEVELS-1) tiles can be added in each dimension. The driver enforces a hard limit of 10.
- **ADD_ALPHA=[YES/NO]:** Defaults to `YES`. Whether an alpha band is added in case of reprojection.

6.17.3 Update

Updating a COG file generally breaks part of the optimizations, but still produces a valid GeoTIFF file. Starting with GDAL 3.8, to avoid undesired loss of the COG characteristics, opening such a file in update mode will be rejected, unless the IGNORE_COG_LAYOUT_BREAK open option is also explicitly set to YES.

Note that a subset of operations are possible when opening a COG file in read-only mode, like metadata edition (including statistics storage), that will be stored in a auxiliary .aux.xml side-car file.

6.17.4 File format details

6.17.4.1 High level

A Cloud optimized GeoTIFF has the following characteristics:

- TIFF or BigTIFF file
- Tiled (512 pixels by default) for imagery, mask and overviews
- Overviews until the maximum dimension of the smallest overview level is lower than 512 pixels.
- Compressed or not
- Pixel interleaving for multi-band dataset
- Optimized layout of TIFF sections to minimize the number of GET requests needed by a reader doing random read access.

6.17.4.2 Low level

A COG file is organized as the following (if using libtiff >= 4.0.11 or GDAL internal libtiff. For other versions, the layout will be different and some of the optimizations will not be available).

- TIFF/BigTIFF header/signature and pointer to first IFD (Image File Directory)
- "ghost area" with COG optimizations (see *Header ghost area*)
- IFD of the full resolution image, followed by TIFF tags values, excluding the TileOffsets and TileByteCounts arrays.
- IFD of the mask of the full resolution image, if present, followed by TIFF tags values, excluding the TileOffsets and TileByteCounts arrays.
- IFD of the first (largest in dimensions) overview level, if present
- ...
- IFD of the last (smallest) overview level, if present
- IFD of the first (largest in dimensions) overview level of the mask, if present
- ...
- IFD of the last (smallest) overview level of the mask, if present
- TileOffsets and TileByteCounts arrays of the above IFDs
- tile data of the smallest overview, if present (with each tile followed by the corresponding tile of mask data, if present), with *leader and trailer bytes*
- ...
- tile data of the largest overview, if present (interleaved with mask data if present)
- tile data of the full resolution image, if present (interleaved with corresponding mask data if present)

6.17.4.3 Header ghost area

To describe the specific layout of COG files, a description of the features used is located at the beginning of the file, so that optimized readers (like GDAL) can use them and take shortcuts. Those features are described as ASCII strings "hidden" just after the 8 first bytes of a ClassicTIFF (or after the 16 first ones for a BigTIFF). That is the first IFD starts just after those strings. It is completely valid to have *ghost* areas like this in a TIFF file, and readers will normally skip over them. So for a COG file with a transparency mask, those strings will be:

```
GDAL_STRUCTURAL_METADATA_SIZE=000174 bytes
LAYOUT=IFDS_BEFORE_DATA
BLOCK_ORDER=ROW_MAJOR
BLOCK_LEADER=SIZE_AS_UINT4
BLOCK_TRAILER=LAST_4_BYTES_REPEATED
KNOWN_INCOMPATIBLE_EDITION=NO
MASK_INTERLEAVED_WITH_IMAGERY=YES
```

Note

- A newline character *n* is used to separate those strings.
- A space character is inserted after the newline following *KNOWN_INCOMPATIBLE_EDITION=NO*
- For a COG without mask, the *MASK_INTERLEAVED_WITH_IMAGERY* item will not be present of course.

The ghost area starts with `GDAL_STRUCTURAL_METADATA_SIZE=XXXXXX bytes\n` (of a fixed size of 43 bytes) where XXXXXX is a 6-digit number indicating the remaining size of the section (that is starting after the linefeed character of this starting line).

- `LAYOUT=IFDS_BEFORE_DATA`: the IFDs are located at the beginning of the file. GDAL will also makes sure that the tile index arrays are written just after the IFDs and before the imagery, so that a first range request of 16 KB will always get all the IFDs
- `BLOCK_ORDER=ROW_MAJOR`: (strile is a contraction of 'strip or tile') the data for tiles is written in increasing tile id order. Future enhancements could possibly implement other layouts.
- `BLOCK_LEADER=SIZE_AS_UINT4`: each tile data is preceded by 4 bytes, in a *ghost* area as well, indicating the real tile size (in little endian order). See *Tile data leader and trailer* for more details.
- `BLOCK_TRAILER=LAST_4_BYTES_REPEATED`: just after the tile data, the last 4 bytes of the tile data are repeated. See *Tile data leader and trailer* for more details.
- `KNOWN_INCOMPATIBLE_EDITION=NO`: when a COG is generated this is always written. If GDAL is then used to modify the COG file, as most of the changes done on an existing COG file, will break the optimized structure, GDAL will change this metadata item to KNOWN_INCOMPATIBLE_EDITION=YES, and issue a warning on writing, and when reopening such file, so that users know they have *broken* their COG file
- `MASK_INTERLEAVED_WITH_IMAGERY=YES`: indicates that mask data immediately follows imagery data. So when reading data at offset=TileOffset[i] - 4 and size=TileOffset[i+1]-TileOffset[i]+4, you'll get a buffer with:
  - leader with imagery tile size (4 bytes)
  - imagery data (starting at TileOffsets[i] and of size TileByteCounts[i])
  - trailer of imagery (4 bytes)
  - leader with mask tilesize (4 bytes)
  - mask data (starting at mask.TileOffsets[i] and of size mask.TileByteCounts[i], but none of them actually need to be read)
  - trailer of mask data (4 bytes)

  This is only written if INTERLEAVE=PIXEL.
- `INTERLEAVE=BAND` or `INTERLEAVE=TILE`: (GDAL >= 3.11) Reflects the value of the INTERLEAVE creation option. Omission implies INTERLEAVE=PIXEL.

> **Note**
>
> The content of the header ghost area can be retrieved by getting the `GDAL_STRUCTURAL_METADATA` metadata item of the `TIFF` metadata domain on the dataset object (with GetMetadataItem())

6.17.4.4 Tile data leader and trailer

Each tile data is immediately preceded by a leader, consisting of a unsigned 4-byte integer, in little endian order, giving the number of bytes of *payload* of the tile data that follows it. This leader is *ghost* in the sense that the TileOffsets[] array does not point to it, but points to the real payload. Hence the offset of the leader is TileOffsets[i]-4.

For INTERLEAVE=PIXEL or INTERLEAVE=TILE, an optimized reader seeing the `BLOCK_LEADER=SIZE_AS_UINT4` metadata item will thus look for TileOffset[i] and TileOffset[i+1] to deduce it must fetch the data starting at offset=TileOffset[i] - 4 and of size=TileOffset[i+1]-TileOffset[i]+4. It then checks the 4 first bytes to see if the size in this leader marker is consistent with TileOffset[i+1]-TileOffset[i]. When there is no mask, they should normally be equal (modulo the size taken by BLOCK_LEADER and BLOCK_TRAILER). In the case where there is a mask and MASK_INTERLEAVED_WITH_IMAGERY=YES, then the tile size indicated

in the leader will be < TileOffset[i+1]-TileOffset[i] since the data for the mask will follow the imagery data (see MASK_INTERLEAVED_WITH_IMAGERY=YES)

For INTERLEAVE=BAND, the above paragraph applies but the successor of tile i is not tile i+1, but tile i+nTilesPerBand.

Each tile data is immediately followed by a trailer, consisting of the repetition of the last 4 bytes of the payload of the tile data. The size of this trailer is *not* included in the TileByteCounts[] array. The purpose of this trailer is forces readers to be able to check if TIFF writers, not aware of those optimizations, have modified the TIFF file in a way that breaks the optimizations. If an optimized reader detects an inconsistency, it can then fallbacks to the regular/slower method of using TileOffsets[i] + TileByteCounts[i].

6.17.5 Examples

6.17.5.1 Example 1: Create a COG with gdalwarp

```
$ gdalwarp src1.tif src2.tif out.tif -of COG
```

6.17.5.2 Example 2: Create a Web Mercator COG with Google Maps–Compatible Tiling and JPEG Compression

```
$ gdal_translate world.tif world_webmerc_cog.tif -of COG -co TILING_
↪SCHEME=GoogleMapsCompatible -co COMPRESS=JPEG
```

6.17.5.3 Example 3: Create a blank GeoTIFF from an existing file

Use *SPARSE_OK* to create an empty file with the same georeferencing and tiling as the input, without physically writing zero-filled tiles to disk. In this example, an input of size `7640 × 8133` produces a ~4 KB file instead of ~242 MB.

Note sparse GeoTIFF files are not widely supported by non-GDAL software.

```
$ gdal raster create --like in.tif out.tif --of COG \
    --co SPARSE_OK=ON \
    --overwrite
```

6.17.6 See Also

- *GTiff -- GeoTIFF File Format* driver
- If your source dataset is an internally tiled geotiff with the desired georeferencing and compression, using cogger (possibly along with gdaladdo to create overviews) will be much faster than the COG driver.

6.18 COSAR -- TerraSAR-X Complex SAR Data Product

Driver short name

COSAR

Driver built-in by default

This driver is built-in by default

This driver provides the capability to read TerraSAR-X complex data. While most users will receive products in GeoTIFF format (representing detected radiation reflected from the targets, or geocoded data), ScanSAR products will be distributed in COSAR format.

Essentially, COSAR is an annotated binary matrix, with each sample held in 4 bytes (16 bits real, 16 bits imaginary) stored with the most significant byte first (Big Endian). Within a COSAR container there are one or more "bursts" which represent individual ScanSAR bursts. Note that if a Stripmap or Spotlight product is held in a COSAR container it is stored in a single burst.

Support for ScanSAR data is currently under way, due to the difficulties in fitting the ScanSAR "burst" identifiers into the GDAL model.

6.18.1 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.18.2 See Also

- DLR Document TX-GS-DD-3307 "Level 1b Product Format Specification."

6.19 CPG -- Convair PolGASP data

Driver short name

CPG

Driver built-in by default

This driver is built-in by default

NOTE: Implemented as cpgdataset.cpp.

6.19.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.20 CPHD - Compensated Phase History Data

Driver short name

CPHD

Added in version 3.13.

GDAL can read CPHD datasets using the *RAWDataset* driver.

6.20.1 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.20.2 Open options

The following open options exist:

- **INCLUDE_XML=[YES/NO]:** Whether to include CPHD XML in the output, default is YES.

6.20.3 Multidimensional API support

The CPHD driver supports the *Multidimensional Raster Data Model* for reading operations.

The driver supports: - reading groups and subgroups - reading multidimensional dense arrays with a numeric data type - reading numeric or string attributes in groups and arrays

6.20.4 See Also

- CPHD v1.1.0 standard
- CPHD v1.1.0 XML schema

6.21 CTG -- USGS LULC Composite Theme Grid

Driver short name

CTG

Driver built-in by default

This driver is built-in by default

This driver can read USGS Land Use and Land Cover (LULC) grids encoded in the Character Composite Theme Grid (CTG) format. Each file is reported as a 6-band dataset of type Int32. The meaning of each band is the following one :

1. Land Use and Land Cover Code
2. Political units Code

3. Census county subdivisions and SMSA tracts Code
4. Hydrologic units Code
5. Federal land ownership Code
6. State land ownership Code

Those files are typically named grid_cell.gz, grid_cell1.gz or grid_cell2.gz on the USGS site.

- Land Use and Land Cover Digital Data (Data Users Guide 4) - PDF version from USGS
- Land Use and Land Cover Digital Data (Data Users Guide 4) - HTML version converted by Ben Discoe
- USGS LULC data at 250K and 100K

NOTE: Implemented as ctgdataset.cpp.

6.21.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.22 DAAS (Airbus DS Intelligence Data As A Service driver)

Driver short name

DAAS

Added in version 3.0.

Build dependencies

libcurl

This driver can connect to the Airbus DS Intelligence Data As A Service API. GDAL/OGR must be built with Curl support in order for the DAAS driver to be compiled.

Orthorectified (with geotransform) and raw (with RPCs) images are supported.

Overviews are supported.

The API is not publicly available but will be released soon. Further information will be found here: https://api.oneatlas.airbus.com/

6.22.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

6.22.2 Dataset name syntax

The nominal syntax to open a datasource is :

```
DAAS:https://example.com/path/to/image/metadata
```

A more minimal syntax can be used:

```
DAAS:
```

provided that the GET_METADATA_URL open option is filled.

6.22.3 Authentication

Access to the API requires an authentication token. There are two methods supported:

- Authentication with an API key and a client id. They must be provided respectively with the *API_KEY* open option (or *GDAL_DAAS_API_KEY* configuration option) and the *CLIENT_ID* open option (or *GDAL_DAAS_CLIENT_ID* configuration option). In that case, the driver will authenticate against the authentication endpoint to get an access token.
- Directly providing the access token with the *ACCESS_TOKEN* open option (or *GDAL_DAAS_ACCESS_TOKEN* configuration option).

In both cases, the *X_FORWARDED_USER* open option (or *GDAL_DAAS_X_FORWARDED_USER* configuration option) can be specified to fill the HTTP X-Forwarded-User header in requests sent to the DAAS service endpoint with the user from which the request originates from.

See https://api.oneatlas.airbus.com/guides/g-authentication/ for further details

6.22.4 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). The following configuration options are available :

- **GDAL_DAAS_API_KEY=value:** Equivalent of *API_KEY* open option.
- **GDAL_DAAS_CLIENT_ID=value:** Equivalent of *CLIENT_ID* open option.
- **GDAL_DAAS_ACCESS_TOKEN=value:** Equivalent of *ACCESS_TOKEN* open option.
- **GDAL_DAAS_X_FORWARDED_USER=value:** Equivalent of *X_FORWARDED_USER* open option.

6.22.5 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are available :

- **GET_METADATA_URL=value:** URL to the GetImageMetadata endpoint. Required if not specified in the connection string.

- **API_KEY=value:** API key for authentication. If specified, must be used together with the *CLIENT_ID* option. Can be specified also through the *GDAL_DAAS_API_KEY* configuration option.
- **CLIENT_ID=value:** Client id for authentication. If specified, must be used together with the *API_KEY* option. Can be specified also through the *GDAL_DAAS_CLIENT_ID* configuration option.
- **ACCESS_TOKEN=value:** Access token. Can be specified also through the *GDAL_DAAS_ACCESS_TOKEN* configuration option. Exclusive of *API_KEY*/*CLIENT_ID*.
- **X_FORWARDED_USER=value:** User from which the request originates from. Can be specified also through the *GDAL_DAAS_X_FORWARDED_USER* configuration option.
- **BLOCK_SIZE=64-8192:** Defaults to `512`. Size of a block in pixels requested to the server.
- **PIXEL_ENCODING=[AUTO/RAW/PNG/JPEG/JPEG2000]:** Defaults to `AUTO`. Format in which pixels are queried:
  - **AUTO**: for 1, 3 or 4-band images of type Byte, resolves to PNG. Otherwise to RAW
  - **RAW**: compatible with all images. Pixels are requested in a uncompressed raw format.
  - **PNG**: compatible with 1, 3 or 4-band images of type Byte
  - **JPEG**: compatible with 1 or 3-band images of type Byte
  - **JPEG2000**: compatible with all images. Requires GDAL to be built with one of its JPEG2000-capable drivers.
- **MASKS=[YES/NO]:** Defaults to `YES`. Whether to expose mask bands.

6.23 DDS -- DirectDraw Surface

Driver short name

DDS

Build dependencies

Crunch Lib

The DirectDraw Surface file format (uses the filename extension DDS), from Microsoft, is a standard for storing data compressed with the lossy S3 Texture Compression (S3TC) algorithm. The DDS format and compression are provided by the crunch library.

Support for reading has been added in GDAL 3.1. Previous versions have write-only support.

The driver supports the following texture formats: DXT1, DXT1A, DXT3 (default), DXT5 and ETC1. You can set the texture format using the creation option FORMAT.

The driver supports the following compression quality: SUPERFAST, FAST, NORMAL (default), BETTER and UBER. You can set the compression quality using the creation option QUALITY.

More information about Crunch Lib (see below for build instructions of a working fork of that repository)

NOTE: Implemented as ddsdataset.cpp.

6.23.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

6.23.2 Build instructions

Building crunch can be a bit difficult. The *build_fixes* branch of https://github.com/rouault/crunch/ includes a CMake build system, as well as a few fixes that are not found in the upstream repository

6.23.2.1 Build crunch

Linux

```
git clone -b build_fixes https://github.com/rouault/crunch
cd crunch
mkdir build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=$HOME/install-crunch -DCMAKE_BUILD_TYPE=Release
make -j8 install
```

Windows

```
git clone -b build_fixes https://github.com/rouault/crunch
cd crunch
mkdir build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=c:\dev\install-crunch -G "Visual Studio 15 2017 Win64"
cmake --build . --config Release --target install
```

6.23.2.2 Build GDAL against crunch

Linux

```
./configure --with-dds=$HOME/install-crunch
```

Windows

In nmake.local, add the following lines:

```
CRUNCH_INC = -Ic:\dev\install-crunch\include\crunch
CRUNCH_LIB = c:\dev\install-crunch\lib\crunch.lib
```

6.24 DERIVED -- Derived subdatasets driver

Driver short name

DERIVED

Driver built-in by default

This driver is built-in by default

This driver allows accessing subdatasets derived from a given dataset. Those derived datasets have the same projection reference, geo-transform and metadata than the original dataset, but derives new pixel values using gdal pixel functions.

6.24.1 Available functions

Available derived datasets are:

- AMPLITUDE: Amplitude of pixels from input bands
- PHASE: Phase of pixels from input bands
- REAL: Real part of pixels from input bands
- IMAG: Imaginary part of pixels from input bands
- CONJ: Conjugate of pixels from input bands
- INTENSITY: Intensity (squared amplitude) of pixels from input bands
- LOGAMPLITUDE: Log10 of amplitude of pixels from input bands

Note: for non-complex data types, only LOGAMPLITUDE will be listed.

A typical use is to directly access amplitude, phase or log-amplitude of any complex dataset.

6.24.2 Accessing derived subdatasets

Derived subdatasets are stored in the DERIVED_SUBDATASETS metadata domain, and can be accessed using the following syntax:

```
DERIVED_SUBDATASET:FUNCTION:dataset_name
```

where function is one of AMPLITUDE, PHASE, REAL, IMAG, CONJ, INTENSITY, LOGAMPLITUDE. So as to ensure numerical precision, all derived subdatasets bands will have Float64 or CFloat64 precision (depending on the function used).

For instance:

```
$ gdalinfo cint_sar.tif
```

```
Driver: GTiff/GeoTIFF
Files: cint_sar.tif
Size is 5, 6
Coordinate System is `'
GCP Projection =
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0],
    UNIT["degree",0.0174532925199433],
    AUTHORITY["EPSG","4326"]]
GCP[  0]: Id=1, Info=
```

(continues on next page)

(continued from previous page)

```
          (-1910.5,-7430.5) -> (297.507,16.368,0)
GCP[  1]: Id=2, Info=
          (588.5,-7430.5) -> (297.938,16.455,0)
GCP[  2]: Id=3, Info=
          (588.5,7363.5) -> (297.824,16.977,0)
GCP[  3]: Id=4, Info=
          (-1910.5,7363.5) -> (297.393,16.89,0)
Metadata:
  AREA_OR_POINT=Area
  CEOS_ACQUISITION_TIME=19970718024119087
  CEOS_ELLIPSOID=GEM6
  CEOS_INC_ANGLE=24.824
  CEOS_LINE_SPACING_METERS=3.9900000
  CEOS_LOGICAL_VOLUME_ID=0001667400297672
  CEOS_PIXEL_SPACING_METERS=7.9040000
  CEOS_PIXEL_TIME_DIR=INCREASE
  CEOS_PLATFORM_HEADING=347.339
  CEOS_PLATFORM_LATITUDE=16.213
  CEOS_PLATFORM_LONGITUDE=-65.311
  CEOS_PROCESSING_AGENCY=ESA
  CEOS_PROCESSING_COUNTRY=ITALY
  CEOS_PROCESSING_FACILITY=ES
  CEOS_SEMI_MAJOR=6378.1440000
  CEOS_SEMI_MINOR=6356.7590000
  CEOS_SENSOR_CLOCK_ANGLE=90.000
  CEOS_SOFTWARE_ID=ERS2-SLC-6.1
  CEOS_TRUE_HEADING=345.5885834
Image Structure Metadata:
  INTERLEAVE=BAND
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0,    6.0)
Upper Right (    5.0,    0.0)
Lower Right (    5.0,    6.0)
Center      (    2.5,    3.0)
Band 1 Block=5x6 Type=CInt16, ColorInterp=Gray
```

```
$ gdalinfo DERIVED_SUBDATASET:LOGAMPLITUDE:cint_sar.tif
```

```
Driver: DERIVED/Derived datasets using VRT pixel functions
Files: cint_sar.tif
Size is 5, 6
Coordinate System is `'
GCP Projection =
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0],
    UNIT["degree",0.0174532925199433],
```

(continues on next page)

(continued from previous page)

```
    AUTHORITY["EPSG","4326"]]
GCP[  0]: Id=1, Info=
          (-1910.5,-7430.5) -> (297.507,16.368,0)
GCP[  1]: Id=2, Info=
          (588.5,-7430.5) -> (297.938,16.455,0)
GCP[  2]: Id=3, Info=
          (588.5,7363.5) -> (297.824,16.977,0)
GCP[  3]: Id=4, Info=
          (-1910.5,7363.5) -> (297.393,16.89,0)
Metadata:
  AREA_OR_POINT=Area
  CEOS_ACQUISITION_TIME=19970718024119087
  CEOS_ELLIPSOID=GEM6
  CEOS_INC_ANGLE=24.824
  CEOS_LINE_SPACING_METERS=3.9900000
  CEOS_LOGICAL_VOLUME_ID=0001667400297672
  CEOS_PIXEL_SPACING_METERS=7.9040000
  CEOS_PIXEL_TIME_DIR=INCREASE
  CEOS_PLATFORM_HEADING=347.339
  CEOS_PLATFORM_LATITUDE=16.213
  CEOS_PLATFORM_LONGITUDE=-65.311
  CEOS_PROCESSING_AGENCY=ESA
  CEOS_PROCESSING_COUNTRY=ITALY
  CEOS_PROCESSING_FACILITY=ES
  CEOS_SEMI_MAJOR=6378.1440000
  CEOS_SEMI_MINOR=6356.7590000
  CEOS_SENSOR_CLOCK_ANGLE=90.000
  CEOS_SOFTWARE_ID=ERS2-SLC-6.1
  CEOS_TRUE_HEADING=345.5885834
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0,    6.0)
Upper Right (    5.0,    0.0)
Lower Right (    5.0,    6.0)
Center      (    2.5,    3.0)
Band 1 Block=5x6 Type=Float64, ColorInterp=Undefined
```

6.24.3 Listing available subdatasets

Available subdatasets are reported in the DERIVED_SUBDATASETS metadata domain. Only functions that make sense will be reported for a given dataset, which means that AMPLITUDE, PHASE, REAL, IMAG, CONJ and INTENSITY will only be reported if the dataset has at least one complex band. Nevertheless, even if not reported, those derived datasets are still reachable with the syntax presented above.

```
$ gdalinfo -mdd DERIVED_SUBDATASETS cint_sar.tif
```

```
Driver: GTiff/GeoTIFF
Files: cint_sar.tif
Size is 5, 6
Coordinate System is `'
GCP Projection =
GEOGCS["WGS 84",
```

(continues on next page)

(continued from previous page)

```
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0],
    UNIT["degree",0.0174532925199433],
    AUTHORITY["EPSG","4326"]]
GCP[  0]: Id=1, Info=
          (-1910.5,-7430.5) -> (297.507,16.368,0)
GCP[  1]: Id=2, Info=
          (588.5,-7430.5) -> (297.938,16.455,0)
GCP[  2]: Id=3, Info=
          (588.5,7363.5) -> (297.824,16.977,0)
GCP[  3]: Id=4, Info=
          (-1910.5,7363.5) -> (297.393,16.89,0)
Metadata:
  AREA_OR_POINT=Area
  CEOS_ACQUISITION_TIME=19970718024119087
  CEOS_ELLIPSOID=GEM6
  CEOS_INC_ANGLE=24.824
  CEOS_LINE_SPACING_METERS=3.9900000
  CEOS_LOGICAL_VOLUME_ID=0001667400297672
  CEOS_PIXEL_SPACING_METERS=7.9040000
  CEOS_PIXEL_TIME_DIR=INCREASE
  CEOS_PLATFORM_HEADING=347.339
  CEOS_PLATFORM_LATITUDE=16.213
  CEOS_PLATFORM_LONGITUDE=-65.311
  CEOS_PROCESSING_AGENCY=ESA
  CEOS_PROCESSING_COUNTRY=ITALY
  CEOS_PROCESSING_FACILITY=ES
  CEOS_SEMI_MAJOR=6378.1440000
  CEOS_SEMI_MINOR=6356.7590000
  CEOS_SENSOR_CLOCK_ANGLE=90.000
  CEOS_SOFTWARE_ID=ERS2-SLC-6.1
  CEOS_TRUE_HEADING=345.5885834
Metadata (DERIVED_SUBDATASETS):
  DERIVED_SUBDATASET_1_NAME=DERIVED_SUBDATASET:AMPLITUDE:cint_sar.tif
  DERIVED_SUBDATASET_1_DESC=Amplitude of input bands from cint_sar.tif
  DERIVED_SUBDATASET_2_NAME=DERIVED_SUBDATASET:PHASE:cint_sar.tif
  DERIVED_SUBDATASET_2_DESC=Phase of input bands from cint_sar.tif
  DERIVED_SUBDATASET_3_NAME=DERIVED_SUBDATASET:REAL:cint_sar.tif
  DERIVED_SUBDATASET_3_DESC=Real part of input bands from cint_sar.tif
  DERIVED_SUBDATASET_4_NAME=DERIVED_SUBDATASET:IMAG:cint_sar.tif
  DERIVED_SUBDATASET_4_DESC=Imaginary part of input bands from cint_sar.tif
  DERIVED_SUBDATASET_5_NAME=DERIVED_SUBDATASET:CONJ:cint_sar.tif
  DERIVED_SUBDATASET_5_DESC=Conjugate of input bands from cint_sar.tif
  DERIVED_SUBDATASET_6_NAME=DERIVED_SUBDATASET:INTENSITY:cint_sar.tif
  DERIVED_SUBDATASET_6_DESC=Intensity (squared amplitude) of input bands from cint_sar.
↪tif
  DERIVED_SUBDATASET_7_NAME=DERIVED_SUBDATASET:LOGAMPLITUDE:cint_sar.tif
  DERIVED_SUBDATASET_7_DESC=log10 of amplitude of input bands from cint_sar.tif
Image Structure Metadata:
```

(continues on next page)

(continued from previous page)

```
  INTERLEAVE=BAND
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0,    6.0)
Upper Right (    5.0,    0.0)
Lower Right (    5.0,    6.0)
Center      (    2.5,    3.0)
Band 1 Block=5x6 Type=CInt16, ColorInterp=Gray
```

6.24.4 See Also:

- *Using Derived Bands part of the GDAL VRT tutorial*

6.25 DIMAP -- Spot DIMAP

Driver short name

DIMAP

Driver built-in by default

This driver is built-in by default

This is a read-only read for Spot DIMAP described images. To use, select the METADATA.DIM (DIMAP 1), VOL_PHR.XML (DIMAP 2) or VOL_PNEO.XML (DIMAP 2 VHR-2020) file in a product directory, or the product directory itself.

The imagery is in a distinct imagery file, often a TIFF file, but the DIMAP dataset handles accessing that file, and attaches geolocation and other metadata to the dataset from the metadata xml file.

The content of the <Spectral_Band_Info> node is reported as metadata at the level of the raster band. Note that the content of the Spectral_Band_Info of the first band is still reported as metadata of the dataset, but this should be considered as a deprecated way of getting this information.

For multi-component products, each component will be exposed in a subdataset.

NOTE: Implemented as dimapdataset.cpp.

6.25.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.26 DOQ1 -- First Generation USGS DOQ

Driver short name

DOQ1

Driver built-in by default

This driver is built-in by default

Support for read access, including reading of an affine georeferencing transform, and capture of the projection string. This format is the old, unlabelled DOQ (Digital Ortho Quad) format from the USGS.

NOTE: Implemented as doq1dataset.cpp.

6.26.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.27 DOQ2 -- New Labelled USGS DOQ

Driver short name

DOQ2

Driver built-in by default

This driver is built-in by default

Support for read access, including reading of an affine georeferencing transform, capture of the projection string and reading of other auxiliary fields as metadata. This format is the new, labelled DOQ (Digital Ortho Quad) format from the USGS.

This driver was implemented by Derrick J Brashear.

NOTE: Implemented as doq2dataset.cpp.

See Also: USGS DOQ Standards

6.27.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.28 DTED -- Military Elevation Data

Driver short name

DTED

Driver built-in by default

This driver is built-in by default

GDAL supports DTED Levels 0, 1, and 2 elevation data for read access. Elevation data is returned as 16 bit signed integer. Appropriate projection and georeferencing information is also returned. A variety of header fields are returned dataset level metadata.

6.28.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.28.1.1 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). This paragraph lists the configuration options that can be set to alter the default behavior of the DTED driver.

- **GDAL_DTED_SINGLE_BLOCK=[TRUE/FALSE]:** Defaults to `FALSE`. Determines whether to read the entire file as a single block. See *Read speed*.
- **REPORT_COMPD_CS=[TRUE/FALSE]:** Defaults to `FALSE`. Can be set to TRUE to avoid stripping the vertical CS of compound CS when reading the SRS of a file.
- **DTED_VERIFY_CHECKSUM=[YES/NO]:** Defaults to `NO`. Determine whether to issue an error/warning if the checksum written do the DTED file does not match its contents. See *Checksum Issues*.
- **DTED_APPLY_PIXEL_IS_POINT=[TRUE/FALSE]:** (GDAL >= 3.1) Defaults to `FALSE`. If `TRUE`, apply a pixel-is-point interpretation to the data when reading the geotransform.
- **DTED_ASSUME_CONFORMANT=[TRUE/FALSE]:** (GDAL >= 3.12) Defaults to `FALSE`. If `TRUE`, assume that the DTED file is conformant to the DTED standard and not in two's complement form. This will cause the DTED driver to skip the check for values less than -16000 and not change them to signed magnitude - assuming they are two's complement before the conversion.

6.28.2 Read Issues

6.28.2.1 Read speed

Elevation data in DTED files are organized per columns. This data organization doesn't fit very well with some scanline oriented algorithms and can cause slowdowns, especially for DTED Level 2 datasets. By defining *`GDAL_DTED_SINGLE_BLOCK=TRUE`*, a whole DTED dataset will be considered as a single block. The first access to the file will be slow, but further accesses will be much quicker. Only use that option if you need to do processing on a whole file.

6.28.2.2 Georeferencing Issues

The DTED specification (MIL-PRF-89020B) states that *horizontal datum shall be the World Geodetic System (WGS 84)*. The vertical datum is defined as EGM96, or EPSG:5773. However, there are still people using old data files georeferenced in WGS 72. A header field indicates the horizontal datum code, so we can detect and handle this situation.

- If the horizontal datum specified in the DTED file is WGS84, the DTED driver will report WGS 84 as SRS.
- If the horizontal datum specified in the DTED file is WGS72, the DTED driver will report WGS 72 as SRS and issue a warning.
- If the horizontal datum specified in the DTED file is neither WGS84 nor WGS72, the DTED driver will report WGS 84 as SRS and issue a warning.

6.28.2.3 Checksum Issues

The default behavior of the DTED driver is to ignore the checksum while reading data from the files. However, you may specify the environment variable `DTED_VERIFY_CHECKSUM=YES` if you want the checksums to be verified. In some cases, the checksum written in the DTED file is wrong (the data producer did a wrong job). This will be reported as a warning. If the checksum written in the DTED file and the checksum computed from the data do not match, an error will be issued.

6.28.3 Creation Issues

The DTED driver does support creating new files, but the input data must be exactly formatted as a Level 0, 1 or 2 cell. That is the size, and bounds must be appropriate for a cell.

6.28.4 See Also

- Implemented as dteddataset.cpp.

6.29 E57: ASTM E57 3D file format (image part)

Added in version 3.13.

Driver short name

E57

Driver built-in by default

but the JPEG or PNG drivers must be available.

This is a read-only driver that reads 2D images potentially included in a ASTM E57 3D file.

It does NOT read the point cloud part of such files. For that, you may use PDAL's E57 reader.

Masks are combined with the JPEG/PNG image and reported as GDAL mask bands.

If a single JPEG or PNG image is included in the file, it is directly reported. Otherwise if there are multiple images, opening the .e57 file will report several subdatasets, whose connection string is of the form `E57:"{filename.e57}":{image_name}`.

The XML descriptor contained in a E57 file can be accessed in the `xml:E57` metadata domain.

Checksums at end of E57 physical pages are not verified.

6.29.1 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.29.2 See Also

- ASTM Standard Specification for 3D Imaging Data Exchange, Version 1.0

6.30 ECRGTOC -- ECRG Table Of Contents (TOC.xml)

Driver short name

ECRGTOC

Driver built-in by default

This driver is built-in by default

This is a read-only reader for ECRG (Enhanced Compressed Raster Graphic) products, that uses the table of content file, TOC.xml, and exposes it as a virtual dataset whose coverage is the set of ECRG frames contained in the table of content.

The driver will report a different subdataset for each subdataset found in the TOC.xml file. Each subdataset consists of the frames of same product id, disk id, and with same scale.

Result of a gdalinfo on a TOC.xml file.

```
Subdatasets:
  SUBDATASET_1_NAME=ECRG_TOC_ENTRY:ECRG:FalconView:1_500_K:ECRG_Sample/EPF/TOC.xml
  SUBDATASET_1_DESC=Product ECRG, Disk FalconView, Scale 1:500 K
```

6.30.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.30.2 See Also

- *NITF -- National Imagery Transmission Format (also CIB, CADRG, ECRG, HRE)*: format of the ECRG frames
- MIL-PRF-32283 : specification of ECRG products

NOTE: Implemented as ecrgtocdataset.cpp

6.31 ECW -- Enhanced Compressed Wavelets (.ecw)

Driver short name

ECW

Build dependencies

ECW SDK

GDAL supports reading and writing ECW files using the ERDAS ECW/JP2 SDK developed by Hexagon Geospatial (formerly Intergraph, ERDAS, ERMapper). Support is optional and requires linking in the libraries available from the ECW/JP2 SDK Download page.

6.31.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.31.2 Licensing

The ERDAS ECW/JP2 SDK v5.x is available under multiple license types. For Desktop usage, decoding any sized ECW/JP2 image is made available free of charge. To compress, deploy on a Server platform, or decode unlimited sized files on Mobile platforms a license must be purchased from Hexagon Geospatial.

6.31.3 History

- v3.x - Last release, 2006
- v4.x - Last release, 2012
- v5.x - Active development, 2013 - current

6.31.4 Creation Options

The ERDAS ECW/JP2 v4.x and v5.x SDK is only free for image decompression. To compress images it is necessary to build with the read/write SDK and to provide an OEM licensing key at runtime which may be purchased from ERDAS.

For those still using the ECW 3.3 SDK, images less than 500MB may be compressed for free, while larger images require licensing from ERDAS. See the licensing agreement and the LARGE_OK option.

Files to be compressed into ECW format must also be at least 128x128. ECW currently only supports 8 bits per channel for ECW Version 2 files. ECW Version 3 files supports 16 bits per channel (as Uint16 data type). Please see Creation options to enable ECW V3 files writing

When writing coordinate system information to ECW files, many less common coordinate systems are not mapped properly. If you know the ECW name for the coordinate system you can force it to be set at creation time with the PROJ and DATUM creation options.

ECW format does not support creation of overviews since the ECW format is already considered to be optimized for "arbitrary overviews".

6.31.5 Creation Options:

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following creation options are supported:

- **LARGE_OK=[YES/NO]:** *(v3.x SDK only)* Allow compressing files larger than 500MB in accordance with EULA terms. Deprecated since v4.x and replaced by *ECW_ENCODE_KEY* & *ECW_ENCODE_COMPANY*.
- **ECW_ENCODE_KEY=value:** *(v4.x SDK or higher)* Provide the OEM encoding key to unlock encoding capability up to the licensed gigapixel limit. The key is approximately 129 hex digits long. The Company and Key must match and must be re-generated with each minor release of the SDK. It may also be provided globally as a configuration option.
- **ECW_ENCODE_COMPANY=value:** *(v4.x SDK or higher)* Provide the name of the company in the issued OEM key (see *ECW_ENCODE_KEY*). The Company and Key must match and must be re-generated with each minor release of the SDK. It may also be provided globally as a configuration option.
- **TARGET=value:** Set the target size reduction as a percentage of the original. If not provided defaults to 90% for greyscale images, and 95% for RGB images.
- **PROJ=value:** Name of the ECW projection string to use. Common examples are NUTM11, or GEODETIC.
- **DATUM=value:** Name of the ECW datum string to use. Common examples are WGS84 or NAD83.
- **UNITS=[METERS/FEET]:** Defaults to `METERS`. Name of the ECW projection units to use : METERS (default) or FEET (us-foot).
- **ECW_FORMAT_VERSION=[2/3]:** Defaults to `2`. When building with the ECW 5.x SDK this option can be set to allow ECW Version 3 files to be created. Default, 2 to retain widest compatibility.

6.31.6 Configuration Options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). The ERDAS ECW SDK supports a variety of configuration options to control various features. See the ECW SDK documentation for full details on the meaning of these options.

- **ECW_CACHE_MAXMEM=<bytes>:** maximum bytes of RAM used for in-memory caching. If not set, up to one quarter of physical RAM will be used by the SDK for in-memory caching.
- **ECWP_CACHE_LOCATION=<path>:** Path to a directory to use for caching ECWP results. If unset ECWP caching will not be enabled.
- **ECWP_CACHE_SIZE_MB=<megabytes>:** The maximum number of megabytes of space in the *ECWP_CACHE_LOCATION* to be used for caching ECWP results.
- **ECWP_BLOCKING_TIME_MS=<milliseconds>:** Defaults to `10000`. time an ecwp:// blocking read will wait before returning.
- **ECWP_REFRESH_TIME_MS=<milliseconds>:** Defaults to `10000`. time delay between blocks arriving and the next refresh callback. For the purposes of GDAL this is the amount of time the driver will wait for more data on an ecwp connection for which the final result has not yet been returned. If set small then RasterIO() requests will often produce low resolution results.
- **ECW_TEXTURE_DITHER=[TRUE/FALSE]:** Defaults to `TRUE`. This may be set to FALSE to disable dithering when decompressing ECW files.
- **ECW_FORCE_FILE_REOPEN=[TRUE/FALSE]:** Defaults to `FALSE`. This may be set to TRUE to force open a file handle for each file for each connection made.
- **ECW_CACHE_MAXOPEN=<integer>:** The maximum number of files to keep open for ECW file handle caching. Defaults to unlimited.

- **ECW_RESILIENT_DECODING=[TRUE/FALSE]:** Defaults to TRUE. Controls whether the reader should be forgiving of errors in a file, trying to return as much data as is available. If set to FALSE an invalid file will result in an error.

The GDAL-specific options:

- **ECW_ALWAYS_UPWARD=[TRUE/FALSE]:** Defaults to TRUE. If TRUE, the driver sets negative Y-resolution and assumes an image always has the "Upward" orientation (Y coordinates increase upward). This may be set to FALSE to let the driver rely on the actual image orientation, using Y-resolution value (sign) of an image, to allow correct processing of rare images with "Downward" orientation (Y coordinates increase "Downward" and Y-resolution is positive). Defaults to TRUE.

6.31.6.1 ECW Version 3 Files

ECW 5.x SDK introduces a new file format version which,

1. Storage of data statistics, histograms, metadata, RPC information within the file header
2. Support for UInt16 data type
3. Ability to update regions within an existing ECW v3 file
4. Introduces other space saving optimizations

Note: This version is not backward compatible and will fail to decode in v3.x or v4.x ECW/JP2 SDK's. The File VERSION Metadata will advertise whether the file is ECW v2 or ECW v3.

6.31.6.2 ECWP

In addition to local files, this driver also supports access to streaming network imagery services using the proprietary "ECWP" protocol from the ERDAS APOLLO product. Use the full ecwp:// prefixed dataset url as input. When built with ECW/JP2 SDK v4.1 or newer it is also possible to take advantage of rfc-24 for asynchronous / progressive streaming access to ECWP services.

6.31.6.3 Metadata / Georeferencing

The PROJ, DATUM and UNITS found in the ECW header are reported in the ECW metadata domain. They can also be set with the SetMetadataItem() method, in order to update the header information of an existing ECW file, opened in update mode, without modifying the imagery.

The geotransform and projection can also be modified with the SetGeoTransform() and SetProjection() methods. If the projection is set with SetProjection() and the PROJ, DATUM or UNITS with SetMetadataItem(), the later values will override the values built from the projection string.

All those can for example be modified with the -a_ullr, -a_srs or -mo switches of the *gdal_edit* utility.

For example:

```
gdal_edit.py -mo DATUM=WGS84 -mo PROJ=GEODETIC -a_ullr 7 47 8 46 test.ecw

gdal_edit.py -a_srs EPSG:3068 -a_ullr 20800 22000 24000 19600 test.ecw
```

6.31.6.4 File Metadata Keys:

- FILE_METADATA_ACQUISITION_DATE
- FILE_METADATA_ACQUISITION_SENSOR_NAME
- FILE_METADATA_ADDRESS
- FILE_METADATA_AUTHOR

- FILE_METADATA_CLASSIFICATION
- FILE_METADATA_COMPANY - should be set to ECW_ENCODE_COMPANY
- FILE_METADATA_COMPRESSION_SOFTWARE - updated during recompression
- FILE_METADATA_COPYRIGHT
- FILE_METADATA_EMAIL
- FILE_METADATA_TELEPHONE
- CLOCKWISE_ROTATION_DEG
- COLORSPACE
- COMPRESSION_DATE
- COMPRESSION_RATE_ACTUAL
- COMPRESSION_RATE_TARGET. This is the percentage of the target compressed file size divided by the uncompressed file size. This is equal to 100 / (100 - TARGET) where TARGET is the value of the TARGET creation option used at file creation, so a COMPRESSION_RATE_TARGET=1 is equivalent to a TARGET=0 (ie no compression), COMPRESSION_RATE_TARGET=5 is equivalent to TARGET=80 (ie dividing uncompressed file size by 5), etc...
- VERSION

6.31.7 Standalone plugin compilation

Added in version 3.10.

While this driver may be built as part of a whole GDAL build, either in libgdal itself, or as a plugin, it is also possible to only build this driver as a plugin, against an already built libgdal.

The version of the GDAL sources used to build the driver must match the version of the libgdal it is built against.

For example, from a "build_georaster" directory under the root of the GDAL source tree:

```
cmake -S ../frmts/ecw -DCMAKE_PREFIX_PATH=/path/to/GDAL_installation_prefix -DECW_ROOT=/
↪path/to/ecw_sdk_root
cmake --build .
```

Note that such a plugin, when used against a libgdal not aware of it, will be systematically loaded at GDAL driver initialization time, and will not benefit from *deferred plugin loading capabilities <rfc-96>*. For that, libgdal itself must be built with the CMake variable GDAL_REGISTER_DRIVER_ECW_FOR_LATER_PLUGIN=ON set.

6.31.8 See Also

- Implemented as ecwdataset.cpp.
- ERDAS ECW/JP2 SDK Read-Only Redistributable available at www.hexagongeospatial.com
- Further product information available in the User Guide
- Support for non-GDAL specific issues should be directed to the Hexagon Geospatial public forum
- Community contributed patches to apply to ECW SDK 3.3 sources

6.32 EEDAI - Google Earth Engine Data API Image

Driver short name

EEDAI

Build dependencies

libcurl

The driver supports read-only operations to access image content, using Google Earth Engine REST API.

6.32.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

6.32.2 Dataset name syntax

The minimal syntax to open a datasource is :

```
EEDAI:[asset][:band_names]
```

where asset is something like projects/earthengine-public/assets/COPERNICUS/S2/20170430T190351_20170430T190351_T10SEG, and band_names a comma separated list of band names (typically indicated by subdatasets on the main image)

6.32.3 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are available :

- **ASSET=<string>:** To specify the asset if not specified in the connection string.
- **BANDS=value:** Comma separated list of band names.
- **PIXEL_ENCODING=[AUTO/PNG/JPEG/AUTO_JPEG_PNG/GEO_TIFF/NPY]:** Format in which to request pixels.
- **BLOCK_SIZE=<integer>:** Defaults to `256`. Size of a GDAL block, which is the minimum unit to query pixels.
- **VSI_PATH_FOR_AUTH=value:** (GDAL >= 3.10) /vsigs/... path onto which a GOOGLE_APPLICATION_CREDENTIALS path specific option is set

6.32.4 Authentication methods

The following authentication methods can be used:

- Authentication Bearer header passed through the *`EEDA_BEARER`* or *`EEDA_BEARER_FILE`* configuration options.
- Service account private key file, through the *`GOOGLE_APPLICATION_CREDENTIALS`* configuration option, or set as a path-specific option whose value is set in the VSI_PATH_FOR_AUTH open option.

- OAuth2 Service Account authentication through the *EEDA_PRIVATE_KEY*/ *EEDA_PRIVATE_KEY_FILE* + *EEDA_CLIENT_EMAIL* configuration options.
- Finally if none of the above method succeeds, the code will check if the current machine is a Google Compute Engine instance, and if so will use the permissions associated to it (using the default service account associated with the VM). To force a machine to be detected as a GCE instance (for example for code running in a container with no access to the boot logs), you can set *CPL_MACHINE_IS_GCE=YES*.

6.32.5 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). The following configuration options are available :

- **EEDA_BEARER=value:** Authentication Bearer value to pass to the API. This option is only useful when the token is computed by external code. The bearer validity is typically one hour from the time where it as been requested.
- **EEDA_BEARER_FILE=<filename>:** Similar to *EEDA_BEARER* option, except than instead of passing the value directly, it is the filename where the value should be read.
- **GOOGLE_APPLICATION_CREDENTIALS=<file.json>:** Service account private key file that contains a private key and client email
- **EEDA_PRIVATE_KEY=value:** RSA private key encoded as a PKCS#8 PEM file, with its header and footer. Used together with *EEDA_CLIENT_EMAIL* to use OAuth2 Service Account authentication. Requires GDAL to be built against libcrypto++ or libssl.
- **EEDA_PRIVATE_KEY_FILE=<filename>:** Similar to *EEDA_PRIVATE_KEY* option, except than instead of passing the value directly, it is the filename where the key should be read.
- **EEDA_CLIENT_EMAIL=value:** email to be specified together with *EEDA_PRIVATE_KEY*/*EEDA_PRIVATE_KEY_FILE* to use OAuth2 Service Account authentication.
- **CPL_MACHINE_IS_GCE=[YES/NO]:** Defaults to `NO`. If `YES`, forces GDAL to consider the current machine to be a a Google Compute Engine instance. May be needed for code running in a container with no access to the boot logs.

6.32.6 Overviews

The driver expose overviews, following a logic of decreasing power of 2 factors, until both dimensions of the smallest overview are lower than 256 pixels.

6.32.7 Subdatasets

When all bands don't have the same georeferencing, resolution, CRS or image dimensions, the driver will expose subdatasets. Each subdataset groups together bands of the same dimension, extent, resolution and CRS.

6.32.8 Metadata

The driver will expose metadata reported in "properties" as dataset-level or band-level metadata.

6.32.9 Pixel encoding

By default (*PIXEL_ENCODING=AUTO*), the driver will request pixels in a format compatible with the number and data types of the bands. The PNG, JPEG and AUTO_JPEG_PNG can only be used with bands of type Byte.

6.32.9.1 Examples

Get metadata on an image:

```
gdalinfo "EEDAI:" -oo ASSET=projects/earthengine-public/assets/COPERNICUS/S2/
↪20170430T190351_20170430T190351_T10SEG --config EEDA_CLIENT_EMAIL "my@email" --config␣
↪EEDA_PRIVATE_KEY_FILE my.pem
```

or

```
gdalinfo "EEDAI:projects/earthengine-public/assets/COPERNICUS/S2/20170430T190351_
↪20170430T190351_T10SEG" --config EEDA_CLIENT_EMAIL "my@email" --config EEDA_PRIVATE_
↪KEY_FILE my.pem
```

6.32.10 See Also

- *Google Earth Engine Data API driver*

6.33 EHdr -- ESRI .hdr Labelled

Driver short name

EHdr

Driver built-in by default

This driver is built-in by default

GDAL supports reading and writing the ESRI .hdr labeling format, often referred to as ESRI BIL format. Eight, sixteen and thirty-two bit integer raster data types are supported as well as 32 bit floating point. Coordinate systems (from a .prj file), and georeferencing are supported. Unrecognized options in the .hdr file are ignored. To open a dataset select the file with the image file (often with the extension .bil). If present .clr color table files are read, but not written. If present, image.rep file will be read to extract the projection system of SpatioCarte Defense 1.0 raster products.

This driver does not always do well differentiating between floating point and integer data. The GDAL extension to the .hdr format to differentiate is to add a field named PIXELTYPE with values of either FLOAT, SIGNEDINT or UNSIGNEDINT. In combination with the NBITS field it is possible to described all variations of pixel types.

eg.

```
ncols 1375
nrows 649
cellsize 0.050401
xllcorner -130.128639
yllcorner 20.166799
nodata_value 9999.000000
nbits 32
pixeltype float
byteorder msbfirst
```

This driver may be sufficient to read GTOPO30 data.

NOTE: Implemented as ehdrdataset.cpp.

6.33.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.33.2 See Also

- ESRI whitepaper: + Extendable Image Formats for ArcView GIS 3.1 and 3.2 (BIL, see p. 5)
- GTOPO30 - Global Topographic Data
- GTOPO30 Documentation
- SpatioCarte Defense 1.0 specification (in French)
- *SRTMHGT Driver*

6.34 EIR -- Erdas Imagine Raw

Driver short name

EIR

Driver built-in by default

This driver is built-in by default

GDAL supports the Erdas Imagine Raw format for read access including 1, 2, 4, 8, 16 and 32bit unsigned integers, 16 and 32bit signed integers and 32 and 64bit complex floating point. Georeferencing is supported.

To open a dataset select the file with the header information. The driver finds the image file from the header information. Erdas documents call the header file the "raw" file and it may have the extension .raw while the image file that contains the actual raw data may have the extension .bl.

NOTE: Implemented as eirdataset.cpp.

6.34.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.35 ENVI -- ENVI .hdr Labelled Raster

Driver short name

ENVI

Driver built-in by default

This driver is built-in by default

GDAL supports some variations of raw raster files with associated ENVI style .hdr files describing the format. To select an existing ENVI raster file select the binary file containing the data (as opposed to the .hdr file), and GDAL will find the .hdr file by replacing the dataset extension with .hdr.

GDAL should support reading bil, bip and bsq interleaved formats, and most pixel types are supported, including 8bit unsigned, 16 and 32bit signed and unsigned integers, 32bit and 64 bit floating point, and 32bit and 64bit complex floating point. There is limited support for recognising map_info keywords with the coordinate system and georeferencing. In particular, UTM and State Plane should work.

All ENVI header fields are stored in the ENVI metadata domain, and all of these can then be written out to the header file.

Creation Options:

- **INTERLEAVE=[BSQ/BIP/BIL]:** Force the generation specified type of interleaving. **BSQ** -- band sequential (default), **BIP** --- data interleaved by pixel, **BIL** -- data interleaved by line. Starting with GDAL 3.5, when copying from a source dataset with multiple bands which advertises a INTERLEAVE metadata item, if the INTERLEAVE creation option is not specified, the source dataset INTERLEAVE will be automatically taken into account. See *Multiband pixel organization (INTERLEAVE metadata item)* for more details.
- **SUFFIX=[REPLACE/ADD]:** Force adding ".hdr" suffix to supplied filename, e.g. if user selects "file.bin" name for output dataset, "file.bin.hdr" header file will be created. By default header file suffix replaces the binary file suffix, e.g. for "file.bin" name "file.hdr" header file will be created.

NOTE: Implemented as envidataset.cpp.

6.35.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.36 ESAT -- Envisat Image Product

Driver short name

ESAT

Driver built-in by default

This driver is built-in by default

GDAL supports the Envisat product format for read access. All sample types are supported. Files with two matching measurement datasets (MDS) are represented as having two bands. Currently all ASAR Level 1 and above products, and some MERIS and AATSR products are supported.

The control points of the GEOLOCATION GRID ADS dataset are read if available, generally giving a good coverage of the dataset. The GCPs are in WGS84.

Virtually all key/value pairs from the MPH and SPH (primary and secondary headers) are copied through as dataset level metadata.

ASAR and MERIS parameters contained in the ADS and GADS records (excluded geolocation ones) can be retrieved as key/value pairs using the "RECORDS" metadata domain.

NOTE: Implemented as envisatdataset.cpp.

See Also: Envisat at ESA.

6.36.1 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.37 ESRIC -- Esri Compact Cache

Driver short name

ESRIC

Driver built-in by default

This driver is built-in by default

Read Esri Compact Cache V2 as a single raster

6.37.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.37.2 Esri Compact Cache V2

A reader for Esri Compact Cache V2. The cache is stored in multiple files located in a specific folder structure. From the point of view of this driver the raster is represented by the file named conf.xml, which resides in the root folder of the cache. The exact content of this XML file is not fully documented by Esri, and is subject to change. This driver uses only a few of the XML fields, as necessary to read the raster.

6.37.3 Esri Tile Package (.tpkx)

Starting from GDAL 3.8, the driver supports reading the Esri Tile Package Specification. A tile package is a compressed file with ".tpkx" extension. It has a simplified structure, containing image tiles stored in Compact Cache V2 storage format and tiling scheme and other metadata stored in a JSON file.

Note

Tile packages with ".tpk" extension, use compact storage V1 format for cache tiles. The spec for this package type is not available and it is not supported by GDAL.

6.37.3.1 Usage examples

If the /path/Layers contains an Esri Compact Cache in V2 format in the normal Web Mercator tile grid, this command will copy the level 2 content into a TIF file.

```
gdal_translate -outsize 1024 1024 path/Layers/conf.xml output.tif
```

To convert a .tpkx file to a GeoTIFF:

```
gdal_translate -outsize 1024 1024 /path/to/my.tpkx output.tif
```

6.37.3.2 Features and Limitations

- Only V2 Compact cache is supported. This format is identified by the value **esriMapCacheStorageModeCompactV2** in the **CacheInfo.TileCacheInfo.StorageFormat** element of the conf.xml file. Other legacy Esri cache formats are not supported.
- Only caches with tiles of 256 by 256 pixels are supported. These are stored as the values of the **CacheInfo.TileCacheInfo.TileRows** and **CacheInfo.TileCacheInfo.TileCols** elements.
- Tiles in the cache can be in different formats, including JPEG and PNG. The most common are 8 bit color or grayscale JPEG and color PNG with or without opacity. These caches will have the value **JPEG**, **PNG** or **MIXED** in the **CacheInfo.TileImageInfo.CacheTileFormat** element. Other tile formats, such as **PNG8** are not supported. The raster will be promoted to RGB or RGBA by this driver, even if the source is grayscale.
- Only caches with 128 by 128 tiles in a bundle file are supported. This value, the linear number of tiles in a bundle, is contained in the **CacheInfo.CacheStorageInfo.PacketSize** element.
- The spatial reference is obtained from the **CacheInfo.TileCacheInfo.SpatialReference.WKT** field. If this field is not present, the default value of EPSG:4326 is assumed.
- As defined in the standard conf.xml file, the cache tile grid has an origin location, stored in the **CacheInfo.TileCacheInfo.TileOrigin** element, which has X and Y components. The cache size is not explicitly defined. The ESRIC format driver will assume a symmetric area around the 0,0 coordinates, in the reference system coordinates. This is true for the standard Web Mercator and GCS reference systems. If this assumption is not valid, the conf.xml file can be modified, adding a non-standard **CacheInfo.TileCacheInfo.TileEnd** element with the appropriate values for the X and Y components, similar to the **TileOrigin**. Note that the grid origin and size is different from the raster data extent. For example, it is common to generate a cache in the standard Web Mercator tile grid, which is defined for the whole Earth, even if data is only available for a small region of the globe.
- A bundled cache has multiple resolution levels, encoded in a series of **CacheInfo.TileCacheInfo.LODInfos.LODInfo** nodes. The **LODInfo** XML nodes are identified by the **LevelID** value, which have to be successive, starting with 0 for the level with the largest value for the **Resolution** and increasing towards the **LODInfo** node with smallest **Resolution** value. The level convention is similar to other Level, Row and Column tile addressing schemes used in WMTS for example. While common, it is not a requirement to have the resolutions of two successive levels differ by a factor of two. Each cache level will be read as an overview, at the appropriate resolution. The resolution values are in spatial reference distance units per pixel.
- Many caches are built level by level from different sources, similar to many web map tile protocols supported by the GDAL WMS driver. This means that the content of a specific level might be from a different source than an adjacent one, or content for a specific level can be missing altogether. This driver will return opaque black when reading areas which do not have tiles in the cache at a given resolution level, even if data does exist at other levels at the same location.
- Starting from GDAL 3.8, the driver can automatically expand the paletted images to RGBA. The same cache may contain tiles with different color representations.

- A cache can exceed the maximum size supported by GDAL, which is INT32_MAX, in either dimension. By default, the driver will return an error when opening such caches unless the following open option is specified: `IGNORE_OVERSIZED_LODS=YES`.

6.37.4 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are available:

- **EXTENT_SOURCE=[FULL_EXTENT/INITIAL_EXTENT/TILING_SCHEME]:** (GDAL >= 3.9.1) Defaults to `FULL_EXTENT`. Which source should be used for the extent of Esri Tile Package (.tpkx) datasets. By default, or if specifying `FULL_EXTENT`, the "fullExtent" element of the root.json file will be selected, which normally covers all tiles with data. `INITIAL_EXTENT` can be specified to use the default initial extent described in the "initialExtent" element of the root.json file. `TILING_SCHEME` will return the extent of the whole tiling scheme, which is typically world coverage.
- **IGNORE_OVERSIZED_LODS=[YES/NO]:** (GDAL >= 3.13) Defaults to `NO`. Whether to ignore LODs whose computed raster size exceeds the maximum size supported by GDAL (INT32_MAX). By default, an error will be returned when attempting to open such datasets. When set to `YES`, such LODs are ignored.

6.37.5 See Also

- Implemented as esric_dataset.cpp.

6.38 ERS -- ERMapper .ERS

Driver short name

ERS

Driver built-in by default

This driver is built-in by default

GDAL supports reading and writing raster files with .ers header files, with some limitations. The .ers ascii format is used by ERMapper for labeling raw data files, as well as for providing extended metadata, and georeferencing for some other file formats. The .ERS format, or variants are also used to hold descriptions of ERMapper algorithms, but these are not supported by GDAL.

The PROJ, DATUM and UNITS values found in the ERS header are reported in the ERS metadata domain.

6.38.1 Driver capabilities

Supports CreateCopy()

This driver supports the *`GDALDriver::CreateCopy()`* operation

Supports Create()

This driver supports the *`GDALDriver::Create()`* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.38.2 Creation Options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following creation options are supported:

- **PIXELTYPE=value:** By setting this to SIGNEDBYTE, a new Byte file can be forced to be written as signed byte. Starting with GDAL 3.7, this option is deprecated and Int8 should rather be used.
- **PROJ=value:** Name of the ERS projection string to use. Common examples are NUTM11, or GEODETIC. If defined, this will override the value computed by SetProjection() or SetGCPs().
- **DATUM=value:** Name of the ERS datum string to use. Common examples are WGS84 or NAD83. If defined, this will override the value computed by SetProjection() or SetGCPs().
- **UNITS=[METERS/FEET]:** Defaults to `METERS`. Name of the ERS projection units to use : METERS (default) or FEET (us-foot). If defined, this will override the value computed by SetProjection() or SetGCPs().

6.38.3 See Also

- Implemented as ersdataset.cpp.

6.39 EXR -- Extended Dynamic Range Image File Format

Added in version 3.1.

Driver short name

EXR

Build dependencies

libopenexr

OpenEXR is a high dynamic range raster file format. The driver supports reading and writing images in that format.

Georeferencing is written as a WKT CRS string and a 3x3 geotransform matrix in EXR header metadata.

"Deep images" are not supported.

6.39.1 Creation Options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following creation options are supported:

- **COMPRESS=[NONE/RLE/ZIPS/ZIP/PIZ/PXR24/B44/B44A/DWAA/DWAB]:** Defaults to `ZIP`. Compression method. Details on the format Wikipedia page
- **PIXEL_TYPE=[HALF/FLOAT/UINT]:** Pixel type used for encoding.
  - `HALF` corresponds to a IEEE-754 16-bit floating point value.
  - `FLOAT` corresponds to a IEEE-754 32-bit floating point value.
  - `UINT` corresponds to a 32-bit unsigned integer value.

  If not specified, the following GDAL data types will be mapped as following:
  - `Byte` ==> HALF
  - `Int16` ==> HALF (potentially lossy)
  - `UInt16` ==> HALF (potentially lossy)
  - `Int32` ==> FLOAT (potentially lossy)
  - `UInt32` ==> UINT
  - `Float32` ==> FLOAT
  - `Float64` ==> FLOAT (generally lossy)
- **TILED=[YES/NO]:** Defaults to `NO`. By default tiled files will be created, unless this option is set to NO. In Create() mode, setting TILED=NO is not possible.
- **BLOCKXSIZE=<integer>:** Defaults to `256`. Sets tile width.
- **BLOCKYSIZE=<integer>:** Defaults to `256`. Sets tile height.
- **OVERVIEWS=[YES/NO]:** Defaults to `NO`. Whether to create overviews. Only compatible with CreateCopy() mode.
- **OVERVIEW_RESAMPLING=[NEAR/AVERAGE/CUBIC/...]:** Defaults to `CUBIC`. Resampling method to use for overview creation.
- **PREVIEW=[YES/NO]:** Defaults to `NO`. Whether to create a preview. Only compatible with CreateCopy() mode, and with RGB(A) data of type Byte.
- **AUTO_RESCALE=[YES/NO]:** Whether to rescale Byte RGB(A) values from 0-255 to the 0-1 range usually used in EXR ecosystem.
- **DWA_COMPRESSION_LEVEL=<integer>:** DWA compression level. The higher, the more compressed the image will be (and the more artifacts). Defaults to 45 for OpenEXR 2.4

6.39.2 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Create()

With the caveat, that it is only for tiled data, and each tile must be written at most once, and written tiles cannot be read back before dataset closing.

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.40 FAST -- EOSAT FAST Format

Driver short name

FAST

Driver built-in by default

This driver is built-in by default

Supported reading from FAST-L7A format (Landsat TM data) and EOSAT Fast Format Rev. C (IRS-1C/1D data). If you want to read other datasets in this format (SPOT), write to me (Andrey Kiselev, dron@ak4719.spb.edu). You should share data samples with me.

Datasets in FAST format represented by several files: one or more administrative headers and one or more files with actual image data in raw format. Administrative files contains different information about scene parameters including filenames of images. You can read files with administrative headers with any text viewer/editor, it is just plain ASCII text.

This driver wants administrative file for input. Filenames of images will be extracted and data will be imported, every file will be interpreted as band.

6.40.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.40.2 Data

6.40.2.1 FAST-L7A

FAST-L7A consists form several files: big ones with image data and three small files with administrative information. You should give to driver one of the administrative files:

- L7fppprrr_rrrYYYYMMDD_HPN.FST: panchromatic band header file with 1 band
- L7fppprrr_rrrYYYYMMDD_HRF.FST: VNIR/ SWIR bands header file with 6 bands
- L7fppprrr_rrrYYYYMMDD_HTM.FST: thermal bands header file with 2 bands

All raw images corresponded to their administrative files will be imported as GDAL bands.

From the `` Level 1 Product Output Files Data Format Control Book'':

```
The file naming convention for the FAST-L7A product files is L7fppprrr_rrrYYYYMMDD_AAA.
FST where L7 = Landsat 7 mission f = ETM+ format (1 or 2) (data not pertaining to a
specific format defaults to 1) ppp = starting path of the product rrr_rrr = starting and
ending rows of the product YYYYMMDD = acquisition date of the image AAA = file type:  HPN
= panchromatic band header file HRF = VNIR/ SWIR bands header file HTM = thermal bands
header file B10 = band 1 B20 = band 2 B30 = band 3 B40 = band 4 B50 = band 5 B61 = band
6L B62 = band 6H B70 = band 7 B80 = band 8 FST = FAST file extension
```

So you should give to driver one of the `L7fppprrr_rrrYYYYMMDD_HPN.FST`, `L7fppprrr_rrrYYYYMMDD_HRF.FST` or `L7fppprrr_rrrYYYYMMDD_HTM.FST` files.

6.40.2.2 IRS-1C/1D

Fast Format REV. C does not contain band filenames in administrative header. So we should guess band filenames, because different data distributors name their files differently. Several naming schemes hardcoded in GDAL's FAST driver. These are:

```
<header>.<ext> <header>.1.<ext> <header>.2.<ext> ...
```

or

```
<header>.<ext> band1.<ext> band2.<ext> ...
```

or

```
<header>.<ext> band1.dat band2.dat ...
```

or

```
<header>.<ext> imagery1.<ext> imagery2.<ext> ...
```

or

```
<header>.<ext> imagery1.dat imagery2.dat ...
```

in lower or upper case. Header file could be named arbitrarily. This should cover majority of distributors fantasy in naming files. But if you out of luck and your datasets named differently you should rename them manually before importing data with GDAL.

GDAL also supports the logic for naming band files for datasets produced by Euromap GmbH for IRS-1C/IRS-1D PAN, LISS3 and WIFS sensors. Their filename logic is explained in the Euromap Naming Conventions document.

6.40.3 Georeference

All USGS projections should be supported (namely UTM, LCC, PS, PC, TM, OM, SOM). Contact me if you have troubles with proper projection extraction.

6.40.4 Metadata

Calibration coefficients for each band reported as metadata items.

- **ACQUISITION_DATE**: First scene acquisition date in yyyyddmm format.
- **SATELLITE**: First scene satellite name.
- **SENSOR**: First scene sensor name.
- **BIASn**: Bias value for the channel **n**.
- **GAINn**: Gain value for the channel **n**.

6.40.5 See Also

Implemented as fastdataset.cpp.

Landsat FAST L7A format description available from http://ltpwww.gsfc.nasa.gov/IAS/htmls/l7_review.html (see ESDIS Level 1 Product Generation System (LPGS) Output Files DFCB, Vol. 5, Book 2)

EOSAT Fast Format REV. C description available from http://www.euromap.de/docs/doc_001.html

6.41 FITS -- Flexible Image Transport System

Driver short name

FITS

Build dependencies

libcfitsio

FITS is a format used mainly by astronomers, but it is a relatively simple format that supports arbitrary image types and multi-spectral images, and so has found its way into GDAL. FITS support is implemented in terms of the standard CFITSIO library, which you must have on your system in order for FITS support to be enabled (see *notes on CFITSIO linking*). Both reading and writing of FITS files is supported.

Starting from version 3.0 georeferencing system support is implemented via the conversion of WCS (World Coordinate System) keywords. Only Latitude - Longitude systems (see the FITS standard document) have been implemented, those for which remote sensing processing is commonly used. As 3D Datum information is missing in FITS/WCS standard, Radii and target bodies are translated using the planetary extension proposed here.

Non-standard header keywords that are present in the FITS file will be copied to the dataset's metadata when the file is opened, for access via GDAL methods. Similarly, non-standard header keywords that the user defines in the dataset's metadata will be written to the FITS file when the GDAL handle is closed.

Note to those familiar with the CFITSIO library: The automatic rescaling of data values, triggered by the presence of the BSCALE and BZERO header keywords in a FITS file, is disabled in GDAL < v3.0. Those header keywords are accessible and updatable via dataset metadata, in the same was as any other header keywords, but they do not affect reading/writing of data values from/to the file. Starting from version 3.0 BZERO and BSCALE

keywords are managed via standard *GDALRasterBand::GetOffset()* / *GDALRasterBand::SetOffset()* and *GDALRasterBand::GetScale()* / *GDALRasterBand::SetScale()* GDAL functions and no more referred as metadata.

6.41.1 Multiple image support

Starting with GDAL 3.2, Multi-Extension FITS (MEF) files that contain one or more extensions following the primary HDU are supported. When more than 2 image HDUs are found, they are reported as subdatasets.

The connection string for a given subdataset/HDU is `FITS:"filename.fits":hdu_number`

6.41.2 Binary table support

Starting with GDAL 3.2, binary tables will be exposed as vector layers (update and creation support from GDAL 3.2.1).

The FITS data types are mapped to OGR data types as the following:

Table 2: Data types

TFORM value	TSCAL, TOFFSET value	Occurrence count
'L' (Logical)	ignored	1
'L' (Logical)	ignored	> 1
'X' (bit)	ignored	each bit mapped to a OGR field
'B' (unsigned byte)	1, 0 (unsigned byte) or 1, -128 (signed byte)	1
'B' (unsigned byte)	1, 0 (unsigned byte) or 1, -128 (signed byte)	> 1
'I' (16 bit signed integer)	1, 0	1
'I' (16 bit integer, interpreted as unsigned)	1, 32768	1
'I' (16 bit signed integer)	other than (1,0) and (1,32768)	1
'I' (16 bit integer)	1, 0	>1
'I' (16 bit integer, interpreted as unsigned)	1, 32768	>1
'I' (16 bit signed integer)	other than (1, 0) and (1, 32768)	> 1
'J' (32 bit signed integer)	1, 0	1
'J' (32 bit integer, interpreted as unsigned)	1, 2147483648	1
'J' (32 bit signed integer)	other than (1, 0) and (1, 2147483648)	1
'J' (32 bit integer)	1, 0	>1
'J' (32 bit integer, interpreted as unsigned)	1, 2147483648	>1
'J' (32 bit signed integer)	other than (1, 0) and (1, 2147483648)	> 1
'K' (64 bit signed integer)	1, 0	1
'K' (64 bit signed integer)	other than (1, 0)	1
'K' (64 bit signed integer)	1, 0	> 1
'K' (64 bit signed integer)	other than (1, 0)	> 1
'A' (character)	ignored	if TFORM='Axxx' and no TDIM header
'A' (character)	ignored	TDIM for 2D field, or variable length ('PA
'E' (single precision floating point)	1, 0	1
'E' (single precision floating point)	other than (1, 0)	1
'E' (single precision floating point)	1, 0	> 1
'E' (single precision floating point)	other than (1, 0)	> 1
'D' (double precision floating point)	any	1
'D' (double precision floating point)	any	> 1
'C' (single precision complex)	any	1
'C' (single precision complex)	any	> 1
'M' (double precision complex)	any	1
'M' (double precision complex)	any	> 1

Fields with a repeat count > 1 expressing fixed size arrays, or fields using array descriptors 'P' and 'Q' for variable length arrays are mapped to OGR OFTxxxxxList data types. The potential 2D structure of such field has no direct equivalence in OGR, so OGR will expose a linear structure. For fixed size arrays, the user can retrieve the value of the TDIMxx header in the layer metadata to recover the dimensionality of the field.

Fields that have TSCAL and/or TZERO headers are automatically scaled and offset to the physical value (only applies to numeric data types)

TNULL headers are used for integer numeric data types and for a single-occurrence field to set a OGR field to NULL.

6.41.3 Layer creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALDatasetCreateLayer()* (C) or *Dataset.CreateLayer* (Python). The following layer creation options are available:

- **REPEAT_{fieldname}=<integer>:** For a given field (substitute {fieldname} by its name) of type IntegerList, Integer64List or RealList, specify a fixed number of elements. Otherwise those fields will be created as variable-length FITS columns, which can have performance impact on creation.
- **COMPUTE_REPEAT=[AT_FIELD_CREATION/AT_FIRST_FEATURE_CREATION]:** Defaults to `AT_FIELD_CREATION`. For fields of type IntegerList, Integer64List or RealList, specifies when they are mapped to a FITS column type. The default is AT_FIELD_CREATION, and implies that they will be created as variable-length FITS columns, unless a *REPEAT_{fieldname}* option is specified. When AT_FIRST_FEATURE_CREATION is specified, the number of elements in the first feature will be taken into account to create fixed-size FITS columns.

When using ogr2ogr or *GDALVectorTranslate()* with a FITS source, the FITS header will be taken into account, in particular to help to determine the FITS data type of target columns.

6.41.4 Examples

- Listing subdatasets in a MEF .fits:

```
$ gdalinfo ../autotest/gdrivers/data/fits/image_in_first_and_second_hdu.fits

Driver: FITS/Flexible Image Transport System
Files: ../autotest/gdrivers/data/fits/image_in_first_and_second_hdu.fits
Size is 512, 512
Metadata:
EXTNAME=FIRST_IMAGE
Subdatasets:
SUBDATASET_1_NAME=FITS:"../autotest/gdrivers/data/fits/image_in_first_and_
↪second_hdu.fits":1
SUBDATASET_1_DESC=HDU 1 (1x2, 1 band), FIRST_IMAGE
SUBDATASET_2_NAME=FITS:"../autotest/gdrivers/data/fits/image_in_first_and_
↪second_hdu.fits":2
SUBDATASET_2_DESC=HDU 2 (1x3, 1 band)
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0,  512.0)
Upper Right (  512.0,    0.0)
Lower Right (  512.0,  512.0)
Center      (  256.0,  256.0)
```

- Opening a given raster HDU:

```
$ gdalinfo FITS:"../autotest/gdrivers/data/fits/image_in_first_and_second_
↪hdu.fits":1

Driver: FITS/Flexible Image Transport System
Files: none associated
Size is 1, 2
Metadata:
EXTNAME=FIRST_IMAGE
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0,    2.0)
Upper Right (    1.0,    0.0)
Lower Right (    1.0,    2.0)
Center      (    0.5,    1.0)
Band 1 Block=1x1 Type=Byte, ColorInterp=Undefined
```

- Listing potential binary tables in a FITS file:

```
$ ogrinfo my.fits
```

- Converting a GeoPackage layer into a FITS binary table:

```
$ ogr2ogr out.fits my.gpkg my_table
```

6.41.5 Other

NOTE: Implemented as fitsdataset.cpp.

6.41.6 Notes on CFITSIO linking in GDAL

6.41.6.1 Linux

From source

Install CFITSIO headers from your distribution (eg, cfitsio-devel on Fedora; libcfitsio-dev on Debian-Ubuntu), then compile GDAL as usual. CFITSIO will be automatically detected and linked.

From distributions

On Fedora/CentOS install CFITSIO then GDAL with dnf (yum): cfitsio is automatically linked.

6.41.6.2 MacOSX

The last versions of the MacOSX packages are not linked against CFITSIO. Install CFITSIO as described in the official documentation.

6.41.7 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.42 GenBin -- Generic Binary (.hdr labelled)

Driver short name

GenBin

Driver built-in by default

This driver is built-in by default

This driver supporting reading "Generic Binary" files labelled with a .hdr file, but distinct from the more common ESRI labelled .hdr format (EHdr driver). The origin of this format is not entirely clear. The .hdr files supported by this driver are look something like this:

```
{{{
BANDS:       1
ROWS:     6542
COLS:     9340
...
}}}
```

Pixel data types of U8, U16, S16, F32, F64, and U1 (bit) are supported. Georeferencing and coordinate system information should be supported when provided.

NOTE: Implemented as genbindataset.cpp.

6.42.1 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.43 Oracle Spatial GeoRaster

Driver short name

GeoRaster

Build dependencies

Oracle client libraries

This driver supports reading and writing raster data in Oracle Spatial GeoRaster format (10g or later). The Oracle Spatial GeoRaster driver is optionally built as a GDAL plugin, but it requires Oracle client libraries.

When opening a GeoRaster, its name should be specified in the form:

georaster:<user>{,/}<pwd>{,@}[db],[schema.][table],[column],[where]
georaster:<user>{,/}<pwd>{,@}[db],<rdt>,<rid>

Where:

user = Oracle server user name, when a proxy user is used, it is in the format of proxyuser[username].
pwd = user password, double quote the password when the password has '@' or ',', or the password is case-sensitive.
db = Oracle Net connect identifier. This can be the TNS alias defined in the tnsnames.ora, or a quoted string of Oracle Net connect descriptor. The location of the tnsnames.ora is configured by TNS_ADMIN environment variable.
schema = name of a schema
table = name of a GeoRaster table (table that contains GeoRaster columns)
column = name of a column data type MDSYS.SDO_GEORASTER
where = a simple where clause to identify one or multiple GeoRaster(s)
rdt = name of a raster data table
rid = numeric identification of one GeoRaster

Examples:

geor:scott,tiger,demodb,table,column,id=1
geor:scott,tiger,demodb,table,column,"id = 1"
geor:gisuser[scott]/welcome,demodb,table,column,"id = 1"
geor:/@demodb,table,column,"id = 1"

geor:scott/\"Welcome@Tiger\"@"(description=(retry_count=20)(retry_delay=3)(address=(protocol=tcps)(port=1522)(host=adb.ca-toronto-1.oraclecloud.com))(connect_data=(service_name=g0eab5a4f6c8694_low.adb.oraclecloud.com))(security=(ssl_server_dn_match=yes)(
"georaster:scott/tiger@demodb,table,column,gain>10"
"georaster:scott/tiger@demodb,table,column,city='Brasilia'"
georaster:scott,tiger,,rdt_10$,10

geor:scott/tiger,,rdt_10$,10

Note: Do not use space around the field values and the commas.

Note: The third example shows that the proxy user `gisuser` is used to connect to user `scott` The password `welcome` is the password to the proxy user `gisuser`

Note: The fourth example shows how to connect when the database user credentials are stored in a client-side Oracle wallet.

Note: The fifth example shows how to double quote the case-sensitive password and the password that contains '@', this example also shows how to use the Oracle Net connect descriptor to connect to an Oracle ADB.

Note: In the last two examples, the Oracle Net connect identifier is left empty (",,") because it is connecting to a local database that is configured in the environment.

Note: If the query results in more than one GeoRaster it will be treated as a GDAL metadata's list of sub-datasets (see below)

6.43.1 Driver capabilities

Supports CreateCopy()

This driver supports the *`GDALDriver::CreateCopy()`* operation

Supports Create()

This driver supports the *`GDALDriver::Create()`* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.43.2 Browsing the database for GeoRasters

By providing some basic information the GeoRaster driver is capable of listing the existing rasters stored on the server:

To list all the GeoRaster table on the server that belongs to that user name and database:

```
gdalinfo georaster:scott/tiger@db1
```

To list all the GeoRaster type columns that exist in that table:

```
gdalinfo georaster:scott/tiger@db1,table_name
```

That will list all the GeoRaster objects stored in that table.

```
gdalinfo georaster:scott/tiger@db1,table_name,georaster_column
```

That will list all the GeoRaster existing on that table according to a Where clause.

```
gdalinfo georaster:scott/tiger@db1,table_name,georaster_column,city='Brasilia'
```

Note that the result of those queries are returned as GDAL metadata sub-datasets, e.g.:

```
gdalinfo georaster:scott/tiger
# Driver: GeoRaster/Oracle Spatial GeoRaster
# Subdatasets:
# SUBDATASET_1_NAME=georaster:scott,tiger,,LANDSAT
# SUBDATASET_1_DESC=Table:LANDSAT
# SUBDATASET_2_NAME=georaster:scott,tiger,,GDAL_IMPORT
# SUBDATASET_2_DESC=Table:GDAL_IMPORT
```

6.43.3 Creation Options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following creation options are supported:

- **BLOCKXSIZE=value:** The number of pixel columns per raster block.
- **BLOCKYSIZE=value:** The number of pixel rows per raster block.
- **BLOCKBSIZE=value:** The number of bands per raster block.
- **BLOCKING=value:** Decline the use of blocking (NO) or request an automatic blocking size (OPTIMAL-PADDING).
- **SRID=value:** Assign a specific EPSG projection/reference system identification to the GeoRaster.
- **INTERLEAVE=[BAND/LINE/PIXEL]:** Band interleaving mode, BAND, LINE, PIXEL (or BSQ, BIL, BIP) for band sequential, Line or Pixel interleaving. Starting with GDAL 3.5, when copying from a source dataset with multiple bands which advertises a INTERLEAVE metadata item, if the INTERLEAVE creation option is not specified, the source dataset INTERLEAVE will be automatically taken into account, unless the *COMPRESS* creation option is specified.
- **DESCRIPTION=value:** A simple description of a newly created table in SQL syntax. If the table already exist, this create option will be ignored, e.g.:

  ```
  gdal_translate -of georaster landsat_823.tif geor:scott/tiger@orcl,landsat,raster \
    -co DESCRIPTION="(ID NUMBER, NAME VARCHAR2(40), RASTER MDSYS.SDO_GEORASTER)" \
    -co INSERT="VALUES (1,'Scene 823',SDO_GEOR.INIT())"
  ```

- **INSERT=value:** A simple SQL insert/values clause to inform the driver what values to fill up when inserting a new row on the table, e.g.:

  ```
  gdal_translate -of georaster landsat_825.tif geor:scott/tiger@orcl,landsat,raster \
    -co INSERT="(ID, RASTER) VALUES (2,SDO_GEOR.INIT())"
  ```

- **GENPYRAMID=value:** Generate pyramid after a GeoRaster object have been loaded to the database. The content of that parameter must be the resampling method of choice NN (nearest neighbor) , BILINEAR, BI-QUADRATIC, CUBIC, AVERAGE4 or AVERAGE16. If *GENPYRLEVELS* is not informed the PL/SQL function sdo_geor.generatePyramid will calculate the number of levels to generate.

- **GENPYRLEVELS=value:** Define the number of pyramid levels to be generated. If *GENPYRAMID* is not informed the resample method NN (nearest neighbor) will apply.
- **GENSTATS=[TRUE/FALSE]:** Defaults to FALSE. To generate statistics from the given rasters, set this value to TRUE. Default value is FALSE. This option must be present in order to generate the stats, otherwise, other GENSTATS options are ignored.
- **GENSTATS_SAMPLINGFACTOR=<integer>:** Defaults to 1. Defines the number of cells skipped in both row and column dimensions when the statistics are computed. For example, when setting this value to 4, one-sixteenth of the cells are sampled. The higher the value, the less accurate the statistics are likely to be, but the more quickly they will be computed. Defaults to 1, which means that all cells are sampled.
- **GENSTATS_SAMPLINGWINDOW=<integer,integer,integer,integer>:** This parameter identifies the upper-left (row, column) and lower-right (row, column) coordinates of a rectangular window, and raster space is assumed. The intersection of the MBR (minimum bounding rectangle) of the GeoRaster object in raster space is used for computing statistics. When this value is not specified, statistics are computed for the entire raster.
- **GENSTATS_HISTOGRAM=[TRUE/FALSE]:** Defaults to FALSE. When this value is set to TRUE, a histogram will be computed and stored. Defaults to FALSE, so a histogram won't be generated.
- **GENSTATS_LAYERNUMBERS=[<integer,integer,...>/<integer>-<integer>]:** Defines the numbers of the layers for which to compute the statistics. This can include numbers, ranges (indicated by hyphens), and commas to separate any combination of those. For example, '1,3-5,7', '1,3,6', '1-6'. If this value is not specified, statistics will be computed for all layers.
- **GENSTATS_USEBIN=[TRUE/FALSE]:** Defaults to TRUE. Defaults to TRUE. Specifies whether or not to use a provided bin function (specified in *GENSTATS_BINFUNCTION*). When this value is set to TRUE, the bin function to be used follows the following sequence: (1) the bin function specified in *GENSTATS_BINFUNCTION*. (2) the bin function specified by the <binFunction> element in the GeoRaster XML metadata. (3) a dynamically generated bin function generated as follows: Min and max are the actual min and max values of the raster Numbins is defined as: * celldepth = 1, numbins = 2. * cellDepth = 2, numbins = 4. * cellDepth = 4, numbins = 8. * cellDepth >= 8, numbins = 256. When this value is set to FALSE, then the bin function to use is the one generated dynamically as previously described.
- **GENSTATS_BINFUNCTION=<integer,integer,integer,integer,integer>:** An array whose element specify the bin type, total number of bins, first bin number, minimum cell value, and maximum cell value. Bin type must be linear (0). When this value is not specified, and *GENSTATS_USEBIN* is TRUE, then the bin function to use is as follows:
  1. A valid function defined in the GeoRaster metadata.
  2. The same bin function generated when *GENSTATS_USEBIN* is FALSE.
- **GENSTATS_NODATA=[TRUE/FALSE]:** Defaults to FALSE. Specifies whether or not to compare each cell values with NODATA values defined in the metadata when computing statistics. When set to TRUE, all pixels with a NODATA value won't be considered. When set to FALSE, NODATA pixels will be considered as regular pixels. A NODATA value is used for cells whose values are either not known or meaningless
- **COMPRESS=[JPEG-F/JP2-F/DEFLATE/NONE]:** Compression options. The JPEG-F options is lossy, meaning that the original pixel values are changed. The JP2-F compression is lossless if *JP2_QUALITY=100*.
- **QUALITY=0-100:** Defaults to 75. Quality compression option for JPEG ranging from 0 to 100.
- **JP2_QUALITY=<float_value,float_value,...>:** Only if *COMPRESS=JP2-f*. Percentage between 0 and 100. A value of 50 means the file will be half-size in comparison to uncompressed data, 33 means 1/3, etc.. Defaults to 25 (unless the dataset is made of a single band with color table, in which case the default quality is 100).
- **JP2_REVERSIBLE=[YES/NO]:** Only if *COMPRESS=JP2-f*. YES means use of reversible 5x3 integer-only filter, NO use of the irreversible DWT 9-7. Defaults to NO (unless the dataset is made of a single band with color table, in which case reversible filter is used).

- **JP2_RESOLUTIONS=<integer>:** Only if *COMPRESS=JP2-f*. Number of resolution levels. Default value is selected such the smallest overview of a tile is no bigger than 128x128.
- **JP2_BLOCKXSIZE=<integer>:** Defaults to 1024. Only if *COMPRESS=JP2-f*. Tile width.
- **JP2_BLOCKYSIZE=<integer>:** Defaults to 1024. Only if *COMPRESS=JP2-f*. Tile height.
- **JP2_PROGRESSION=[LRCP/RLCP/RPCL/PCRL/CPRL]:** Defaults to LRCP. Only if *COMPRESS=JP2-f*. Progression order.
- **NBITS=[1/2/4]:** Sub byte data type.
- **SPATIALEXTENT=[TRUE/FALSE]:** Defaults to TRUE. Generate Spatial Extents. The default value is TRUE, which means that this option only need to be set to force the Spatial Extent to remain as NULL. If *EXTENTSRID* is not set the Spatial Extent geometry will be generated with the same SRID as the GeoRaster object.
- **EXTENTSRID=value:** SRID code to be used on the Spatial Extent geometry. If the table/column has already a spatial index, the value specified should be the same as the SRID on the Spatial Extents of the other existing GeoRaster objects, on which the spatial index is built.
- **OBJECTTABLE=[TRUE/FALSE]:** Defaults to FALSE. To create RDT as SDO_RASTER object set to TRUE. Otherwise, the RDT will be created as regular relational tables. That does not apply for Oracle version older than 11.
- **POOL=[TRUE/FALSE]:** Defaults to FALSE. When this option is set to TRUE, OCI session pool is created and associated with the GeoRaster driver. The session pool is destroyed when the GeoRaster driver is destroyed. This option can be used to improve the performance of the GeoRaster driver in GDAL APIs.
- **POOL_SESSMIN=value:** Defaults to 1. Only if *POOL=TRUE*. The number of the sessions are started when a session pool is created.
- **POOL_SESSMAX=value:** Defaults to 10. Only if *POOL=TRUE*. The maximum number of the sessions that can be opened in a session pool. Must be greater than 0.
- **POOL_SESSINCR=value:** Defaults to 2. Only if *POOL=TRUE*. The next increment number for the sessions to be started if the current number of sessions is less than POOL_SESSMAX. Must be greater than 0.

6.43.4 Open Options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are supported:

- **POOL=[TRUE/FALSE]:** Defaults to FALSE. When this option is set to TRUE, OCI session pool is created and associated with the GeoRaster driver. The session pool is destroyed when the GeoRaster driver is destroyed. This option can be used to improve the performance of the GeoRaster driver when there are concurrent reading through the GeoRaster driver in GDAL APIs.
- **POOL_SESSMIN=value:** Defaults to 1. Only if *POOL=TRUE*. The number of the sessions are started when a session pool is created.
- **POOL_SESSMAX=value:** Defaults to 10. Only if *POOL=TRUE*. The maximum number of the sessions that can be opened in a session pool. Must be greater than 0.
- **POOL_SESSINCR=value:** Defaults to 2. Only if *POOL=TRUE*. The next increment number for the sessions to be started if the current number of sessions is less than POOL_SESSMAX. Must be greater than 0.

6.43.5 Importing GeoRaster

During the process of importing raster into a GeoRaster object it is possible to give the driver a simple SQL table definition and also a SQL insert/values clause to inform the driver about the table to be created and the values to be added to the newly created row. The following example does that:

```
gdal_translate -of georaster Newpor.tif georaster:scott/tiger,,landsat,scene \
  -co "DESCRIPTION=(ID NUMBER, SITE VARCHAR2(45), SCENE MDSYS.SDO_GEORASTER)" \
  -co "INSERT=VALUES(1,'West fields', SDO_GEOR.INIT())" \
  -co "BLOCKXSIZE=512" -co "BLOCKYSIZE=512" -co "BLOCKBSIZE=3" \
  -co "INTERLEAVE=PIXEL" -co "COMPRESS=JPEG-F"
```

Note that the create option DESCRIPTION requires to inform table name (in bold). And column name (underlined) should match the description:

```
gdal_translate -of georaster landsat_1.tif georaster:scott/tiger,,landsat,scene \
  -co "DESCRIPTION=(ID NUMBER, SITE VARCHAR2(45), SCENE MDSYS.SDO_GEORASTER)" \
  -co "INSERT=VALUES(1,'West fields', SDO_GEOR.INIT())"
```

If the table "landsat" exist, the option "DESCRIPTION" is ignored. The driver can only update one GeoRaster column per run of gdal_translate. Oracle create default names and values for RDT and RID during the initialization of the SDO_GEORASTER object but user are also able to specify a name and value of their choice.

```
gdal_translate -of georaster landsat_1.tif georaster:scott/tiger,,landsat,scene \
  -co "INSERT=VALUES(10,'Main building', SDO_GEOR.INIT('RDT', 10))"
```

If no information is given about where to store the raster the driver will create (if doesn't exist already) a default table named GDAL_IMPORT with just one GeoRaster column named RASTER and a table GDAL_RDT as the RDT, the RID will be given automatically by the server, example:

```
gdal_translate -of georaster input.tif “geor:scott/tiger@dbdemo”
```

6.43.6 Exporting GeoRaster

A GeoRaster can be identified by a Where clause or by a pair of RDT & RID:

```
gdal_translate -of gtiff geor:scott/tiger@dbdemo,landsat,scene,id=54 output.tif
gdal_translate -of gtiff geor:scott/tiger@dbdemo,st_rdt_1,130 output.tif
```

6.43.7 Cross schema access

As long as the user was granted full access the GeoRaster table and the Raster Data Table, e.g.:

```
% sqlplus scott/tiger
SQL> grant select,insert,update,delete on gdal_import to spock;
SQL> grant select,insert,update,delete on gdal_rdt to spock;
```

It is possible to an user access to extract and load GeoRaster from another user/schema by informing the schema name as showed here:

Browsing:

```
gdalinfo geor:spock/lion@orcl,scott.
gdalinfo geor:spock/lion@orcl,scott.gdal_import,raster,"t.raster.rasterid > 100"
gdalinfo geor:spock/lion@orcl,scott.gdal_import,raster,t.raster.rasterid=101
```

Extracting:

```
gdal_translate geor:spock/lion@orcl,scott.gdal_import,raster,t.raster.rasterid=101out.tif
gdal_translate geor:spock/lion@orcl,gdal_rdt,101 out.tif
```

Note: On the above example that accessing by RDT/RID doesn't need same name as long as the users is granted full access to both tables.

Loading:

```
gdal_translate -of georaster input.tif geor:spock/lion@orcl,scott.
gdal_translate -of georaster input.tif geor:spock/lion@orcl,scott.cities,image \
  -co INSERT="(1,'Rio de Janeiro',sdo_geor.init('cities_rdt'))"
```

6.43.8 General use of GeoRaster

GeoRaster can be used in any GDAL command line tool with all the available options. Like a image subset extraction or re-project:

```
 % gdal_translate -of gtiff geor:scott/tiger@dbdemo,landsat,scene,id=54 output.tif \
 -srcwin 0 0 800 600
 % gdalwarp -of png geor:scott/tiger@dbdemo,st_rdt_1,130 output.png
-t_srs EPSG:9000913
```

Two different GeoRaster can be used as input and output on the same operation:

```
% gdal_translate -of georaster geor:scott/tiger@dbdemo,landsat,scene,id=54 \
geor:scott/tiger@proj1,projview,image -co INSERT="VALUES (102, SDO_GEOR.INIT())"
```

Applications that use GDAL can theoretically read and write from GeoRaster just like any other format but most of then are more inclined to try to access files on the file system so one alternative is to create VRT to represent the GeoRaster description, e.g.:

```
% gdal_translate -of VRT geor:scott/tiger@dbdemo,landsat,scene,id=54 view_54.vrt
% openenv view_54.vrt
```

6.43.8.1 Standalone plugin compilation

Added in version 3.10.

While this driver may be built as part of a whole GDAL build, either in libgdal itself, or as a plugin, it is also possible to only build this driver as a plugin, against an already built libgdal.

The version of the GDAL sources used to build the driver must match the version of the libgdal it is built against.

For example, from a "build_georaster" directory under the root of the GDAL source tree:

```
cmake -S ../frmts/georaster -DCMAKE_PREFIX_PATH=/path/to/GDAL_installation_prefix -
↪DOracle_ROOT=/path/to/instantclient_sdk_root
cmake --build .
```

Note that such a plugin, when used against a libgdal not aware of it, will be systematically loaded at GDAL driver initialization time, and will not benefit from *deferred plugin loading capabilities <rfc-96>*. For that, libgdal itself must be built with the CMake variable GDAL_REGISTER_DRIVER_GEOR_FOR_LATER_PLUGIN=ON set.

6.44 GFF -- Sandia National Laboratories GSAT File Format

Driver short name

GFF

Driver built-in by default

This driver is built-in by default

This read-only GDAL driver is designed to provide access to processed data from Sandia National Laboratories' various experimental sensors. The format is essentially an arbitrary length header containing instrument configuration and performance parameters along with a binary matrix of 16- or 32-bit complex or byte real data.

The GFF format was implemented based on the Matlab code provided by Sandia to read the data. The driver supports all types of data (16-bit or 32-bit complex, real bytes) theoretically, however due to a lack of data only 32-bit complex data has been tested.

Sandia provides some sample data at http://www.sandia.gov/radar/complex-data/.

The extension for GFF formats is .gff.

NOTE: Implemented as gff_dataset.cpp.

6.44.1 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.45 GIF -- Graphics Interchange Format

Driver short name

GIF

Build dependencies

(internal GIF library provided)

GDAL supports reading and writing of normal, and interlaced GIF files. Gif files always appear as having one colormapped eight bit band. GIF files have no support for georeferencing.

A GIF image with transparency will have that entry marked as having an alpha value of 0.0 (transparent). Also, the transparent value will be returned as the NoData value for the band.

If an ESRI *world file* exists with the .gfw, .gifw or .wld extension, it will be read and used to establish the geotransform for the image.

XMP metadata can be extracted from the file, and will be stored as XML raw content in the xml:XMP metadata domain.

6.45.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.45.2 Creation Issues

GIF files can only be created as 1 8bit band using the "CreateCopy" mechanism. If written from a file that is not colormapped, a default greyscale colormap is generated. Transparent GIFs are not currently supported on creation.

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following creation options are supported:

- **WORLDFILE=[ON/OFF]:** Defaults to `OFF`. Force the generation of an associated ESRI world file (.wld). See *World Files* section for details.
- **INTERLACING=[ON/OFF]:** Defaults to `OFF`. Generate an interlaced (progressive) GIF file.

GDAL's internal GIF support is implemented based on source from the giflib 4.1.6 library (written by Gershon Elbor, Eric Raymond and Toshio Kuratomi), hence generating LZW compressed GIF.

The driver was written with the financial support of the DM Solutions Group, and CIET International.

6.45.3 See Also

- giflib Home Page

6.46 GPKG -- GeoPackage raster

Driver short name

GPKG

Build dependencies

libsqlite3 (and any or all of PNG, JPEG, WEBP drivers)

This driver implements full read/creation/update of tables containing raster tiles in the OGC GeoPackage format standard. The GeoPackage standard uses a SQLite database file as a generic container, and the standard defines:

- Expected metadata tables (`gpkg_contents`, `gpkg_spatial_ref_sys`, `gpkg_tile_matrix`, `gpkg_tile_matrix_set`, ...)
- Tile format encoding (PNG and JPEG for base specification, WebP as extension) and tiling conventions
- Naming and conventions for extensions

This driver reads and writes SQLite files from the file system, so it must be run by a user with read/write access to the files it is working with.

The driver can also handle GeoPackage vectors. See *GeoPackage vector* documentation page

Various kind of input datasets can be converted to GeoPackage raster :

- Single band grey level

- Single band with R,G,B or R,G,B,A color table
- Two bands: first band with grey level, second band with alpha channel
- Three bands: Red, Green, Blue
- Four band: Red, Green, Blue, Alpha

GeoPackage rasters only support Byte data type.

All raster extensions standardized by the GeoPackage specification are supported in read and creation :

- *gpkg_webp*: when storing WebP tiles, provided that GDAL is compiled against libwebp.
- *gpkg_zoom_other*: when resolution of consecutive zoom levels does not vary with a factor of 2.

Validation whether a GeoPackage file conforms to the GeoPackage specification can be done with *gdal driver gpkg validate*.

6.46.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.46.2 Opening options

By default, the driver will expose a GeoPackage dataset as a four band (Red,Green, Blue,Alpha) dataset, which gives the maximum compatibility with the various encodings of tiles that can be stored. It is possible to specify an explicit number of bands with the BAND_COUNT opening option. Starting with GDAL 3.6, a special metadata item is written when creating a GeoPackage raster with GDAL (when using the default CUSTOM tiling scheme), and it is used on reading as the default number of bands.

The driver will use the geographic/projected extent indicated in the gpkg_contents table, and do necessary clipping, if needed, to respect that extent. However that information being optional, if omitted, the driver will use the extent provided by the gpkg_tile_matrix_set, which covers the extent at all zoom levels. The user can also specify the USE_TILE_EXTENT=YES open option to use the actual extent of tiles at the maximum zoom level. Or it can specify any of MINX/MINY/MAXX/MAXY to have a custom extent.

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to `GDALOpenEx()` (C) or `gdal.OpenEx` (Python). The following open options are available:

- **TABLE=<table_name>:** Name of the table containing the tiles (called "Tile Pyramid User Data Table" in the GeoPackage specification language). If the GeoPackage dataset only contains one table, this option is not necessary. Otherwise, it is required.
- **ZOOM_LEVEL=value:** Integer value between 0 and the maximum filled in the *gpkg_tile_matrix* table. By default, the driver will select the maximum zoom level, such as at least one tile at that zoom level is found in the raster table.
- **BAND_COUNT=[AUTO/1/2/3/4]:** Number of bands of the dataset exposed after opening. Only used for Byte data type. Some conversions will be done when possible and implemented, but this might fail in some cases, depending on the BAND_COUNT value and the number of bands of the tile. Before GDAL 3.6, the default value is 4 (which is the always safe value). Starting with GDAL 3.6, when the metadata of the file contains an hint of the number of bands, this one is used in AUTO mode (default value), or fallback to 4 when it is not present.
- **MINX=value:** Minimum longitude/easting of the area of interest.
- **MINY=value:** Minimum latitude/northing of the area of interest.
- **MAXX=value:** Maximum longitude/easting of the area of interest.
- **MAXY=value:** Maximum latitude/northing of the area of interest.
- **USE_TILE_EXTENT=[YES/NO]:** Defaults to `NO`. Whether to use the extent of actual existing tiles at the zoom level of the full resolution dataset.
- **TILE_FORMAT=[PNG_JPEG/PNG/PNG8/JPEG/WEBP]:** Format used to store tiles. See *Tile formats*. Only used in update mode and for Byte data type. Defaults to PNG_JPEG, unless, starting with GDAL 3.6, if the raster has one band, in which case PNG is used.
- **QUALITY=1-100:** Defaults to `75`. Quality setting for JPEG and WEBP compression. Only used in update mode.
- **ZLEVEL=1-9:** Defaults to `6`. DEFLATE compression level for PNG tiles. Only used in update mode.
- **DITHER=[YES/NO]:** Defaults to `NO`. Whether to use Floyd-Steinberg dithering (for *`TILE_FORMAT=PNG8`*). Only used in update mode.

Note: open options are typically specified with "-oo name=value" syntax in most GDAL utilities, or with the GDALOpenEx() API call.

6.46.3 Creation issues

Depending of the number of bands of the input dataset and the tile format selected, the driver will do the necessary conversions to be compatible with the tile format.

To add several tile tables to a GeoPackage dataset (seen as GDAL subdatasets), or to add a tile table to an existing vector-only GeoPackage, the generic APPEND_SUBDATASET=YES creation option must be provided.

Fully transparent tiles will not be written to the database, as allowed by the format.

The driver implements the Create() and IWriteBlock() methods, so that arbitrary writing of raster blocks is possible, enabling the direct use of GeoPackage as the output dataset of utilities such as gdalwarp.

On creation, raster blocks can be written only if the geotransformation matrix has been set with SetGeoTransform() This is effectively needed to determine the zoom level of the full resolution dataset based on the pixel resolution, dataset and tile dimensions.

Technical/implementation note: when a dataset is opened with a non-default area of interest (i.e. use of MINX,MINY,MAXX,MAXY or USE_TILE_EXTENT open option), or when creating/ opening a dataset with a non-custom tiling scheme, it is possible that GDAL blocks do not exactly match a single GeoPackage tile. In which case, each GDAL block will overlap four GeoPackage tiles. This is easily handled on the read side, but on creation/update side, such configuration could cause numerous decompression/ recompression of tiles to be done, which might cause

unnecessary quality loss when using lossy compression (JPEG, WebP). To avoid that, the driver will create a temporary database next to the main GeoPackage file to store partial GeoPackage tiles in a lossless (and uncompressed) way. Once a tile has received data for its four quadrants and for all the bands (or the dataset is closed or explicitly flushed with FlushCache()), those uncompressed tiles are definitely transferred to the GeoPackage file with the appropriate compression. All of this is transparent to the user of GDAL API/utilities

The driver updates the GeoPackage `last_change` timestamp when the file is created or modified. If consistent binary output is required for reproducibility, the timestamp can be forced to a specific value by setting the *OGR_CURRENT_DATE* global configuration option. When setting the option, take care to meet the specific time format requirement of the GeoPackage standard, e.g. for version 1.2.

6.46.3.1 Tile formats

Tiled rasters

This section only applies for raster of Byte data type.

GeoPackage can store tiles in different formats, PNG and/or JPEG for the baseline specification, and WebP for extended GeoPackage. Support for those tile formats depend if the underlying drivers are available in GDAL, which is generally the case for PNG and JPEG, but not necessarily for WebP since it requires GDAL to be compiled against the optional libwebp.

By default, GDAL will use a mix of PNG and JPEG tiles (PNG_JPEG tile format, or AUTO). PNG tiles will be used to store tiles that are not completely opaque, either because input dataset has an alpha channel with non fully opaque content, or because tiles are partial due to clipping at the right or bottom edges of the raster, or when a dataset is opened with a non-default area of interest, or with a non-custom tiling scheme. On the contrary, for fully opaque tiles, JPEG format will be used. Starting with GDAL 3.6, if the raster has only one band, the default is PNG.

It is possible to select one unique tile format by setting the creation/open option TILE_FORMAT to one of PNG, JPEG or WEBP. When using JPEG, the alpha channel will not be stored. When using WebP, the gpkg_webp extension will be registered. The lossy compression of WebP is used. Note that a recent enough libwebp (>=0.1.4) must be used to support alpha channel in WebP tiles.

PNG8 can be selected to use 8-bit PNG with a color table up to 256 colors. On creation, an optimized color table is computed for each tile. The DITHER option can be set to YES to use Floyd/Steinberg dithering algorithm, which spreads the quantization error on neighbouring pixels for better rendering (note however than when zooming in, this can cause non desirable visual artifacts). Setting it to YES will generally cause less effective compression. Note that at that time, such an 8-bit PNG formulation is only used for fully opaque tiles, as the median-cut algorithm currently implemented to compute the optimal color table does not support alpha channel (even if PNG8 format would potentially allow color table with transparency). So when selecting PNG8, non fully opaque tiles will be stored as 32-bit PNG.

Tiled gridded coverage data

tiled gridded coverage data can be stored using PNG unsigned 16bit tiles (with potential offset and scaling so as to be able to represent floating point data) or TIFF 32-bit floating-point LZW compressed tiles.

When converting a GDAL Int16 or UInt16 dataset, PNG tiles will be used. When converting a GDAL Float32 dataset, TIFF tiles will be used by default, unless PNG is explicitly selected, in which case scaling and offsetting will be automatically computed for each tile.

6.46.3.2 Tiling schemes

By default, conversion to GeoPackage will create a custom tiling scheme, such that the input dataset can be losslessly converted, both at the pixel and georeferencing level (if using a lossless tile format such as PNG). That tiling scheme is such that its origin (*min_x*, *max_y*) in the gpkg_tile_matrix_set table perfectly matches the top left corner of the dataset, and the selected resolution (*pixel_x_size*, *pixel_y_size*) at the computed maximum zoom_level of the gpkg_tile_matrix table will match the pixel width and height of the raster.

However to ease interoperability with other implementations, and enable use of GeoPackage with tile servicing software, it is possible to select a predefined tiling scheme that has world coverage. The available tiling schemes are :

- *GoogleMapsCompatible*, as described in WMTS 1.0 specification, Annex E.4. That tiling schemes consists of a single 256x256 tile at its zoom level 0, in EPSG:3857 CRS, with extent in easting and northing in the range [-20037508.34,20037508.34].
- *InspireCRS84Quad*, as described in Inspire View Services. That tiling schemes consists of two 256x256 tiles at its zoom level 0, in EPSG:4326 CRS, with extent in longitude in the range [-180,180] and in latitude in the range [-90,90].
- *PseudoTMS_GlobalGeodetic*, based on the global-geodetic profile of OSGeo TMS (Tile Map Service) specification. This has exactly the same definition as *InspireCRS84Quad* tiling scheme. Note however that full interoperability with TMS is not possible due to the origin of numbering of tiles being the top left corner in GeoPackage (consistently with WMTS convention), whereas TMS uses the bottom left corner as origin.
- *PseudoTMS_GlobalMercator*, based on the global-mercator profile of OSGeo TMS (Tile Map Service) specification. That tiling schemes consists of four 256x256 tiles at its zoom level 0, in EPSG:3857 CRS, with extent extent in easting and northing in the range [-20037508.34,20037508.34]. The same remark as with PseudoTMS_GlobalGeodetic applies regarding interoperability with TMS.
- *GoogleCRS84Quad*, as described in OGC 07-057r7 WMTS 1.0 specification, Annex E.3. That tiling schemes consists of a single 256x256 tile at its zoom level 0, in EPSG:4326 CRS, with extent in longitude and latitude in the range [-180,180]. Consequently, at zoom level 0, 64 lines are unused at the top and bottom of that tile. This may cause issues with some implementations of the specification, and there are some ambiguities about the exact definition of this tiling scheme. Using InspireCRS84Quad/PseudoTMS_GlobalGeodetic instead is therefore recommended. NOTE: OGC WMTS Simple Profile 13-082r2 changed the definition of GoogleCRS84Quad (so not implemented by the driver). The new definition includes a -1 level (that cannot be modeled in GeoPackage given constraints on zoom_level being positive or 0), with a single tile at origin -180,90 and whose bottom 128 lines are empty. Levels 0 or greater are identical to the InspireCRS84Quad tiling scheme. So for practical purposes, InspireCRS84Quad in GeoPackage is conformant to the new GoogleCRS84Quad definition.

In all the above tiling schemes, consecutive zoom levels defer by a resolution of a factor of two.

Starting with GDAL 3.2, it is also possible to use a Tile Matrix Set definition, encoded as a JSON file, according to the OGC Two Dimensional Tile Matrix Set standard Examples of such files can be found at http://schemas.opengis.net/tms/1.0/json/examples/ The GDAL data directory also contains files prefixed with `tms_` and with a `.json` extension. If there is a `tms_FOO.json` file, then `FOO` can be used as the value of the TILING_SCHEME creation option. There are restrictions on the types of tile matrix set supported:

- all zoom levels must have the same origin
- consecutive zoom levels defer by a resolution of a factor of two.
- all zoom levels must have the same tile dimension
- variable matrix width tile set are not supported.

6.46.3.3 Nodata value

The concept of the nodata value is only supported for tiled gridded elevation datasets. For regular tiled rasters, the alpha band must rather be used.

For Float32 datasets with TIFF tiles, the concepts of nodata in GDAL and null_value in the GeoPackage internals perfectly match.

For Int16, UInt16 or Float32 with PNG tiles, GDAL will generally remap the input nodata value to another value.

On writing, for PNG tiles, the behavior is the following one:

GDAL data type	Input GDAL nodata value	null_value in GPKG gpkg_2d_gridded_coverage_ancillary
Int16	Any	65535
UInt16	X (if coverage offset == 0 and coverage scale == 1)	X
Float32	Any	65535

On reading, for PNG tiles, the behavior is the following one:

GDAL data type	null_value in GPKG gpkg_2d_gridded_coverage_ancillary	Exposed GDAL nodata value
Int16	>= 32768	-32768
Int16	X <= 32767	X
UInt16	X	X
Float32	X	X

Thus, perfect roundtripping is achieved in the following cases:

GDAL data type	GDAL nodata value	null_value in GPKG gpkg_2d_gridded_coverage_ancillary
Int16	-32768	65535
UInt16	X (if coverage offset == 0 and coverage scale == 1)	X
Float32	65535	65535

6.46.3.4 Creation options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following creation options are available:

- **RASTER_TABLE=value:** Name of tile user table. By default, based on the filename (i.e. if filename is foo.gpkg, the table will be called "foo").
- **APPEND_SUBDATASET=[YES/NO]:** Defaults to `NO`. If set to YES, an existing GeoPackage will not be priorly destroyed, such as to be able to add new content to it.
- **RASTER_IDENTIFIER=value:** Human-readable identifier (e.g. short name), put in the *identifier* column of the *gpkg_contents* table.
- **RASTER_DESCRIPTION=value:** Human-readable description, put in the *description* column of the *gpkg_contents* table.
- **BLOCKSIZE=<integer>:** Defaults to `256`. Block size in width and height in pixels. Maximum supported is 4096. Should not be set when using a non-custom *`TILING_SCHEME`*.
- **BLOCKXSIZE=<integer>:** Defaults to `256`. Block width in pixels. Maximum supported is 4096.
- **BLOCKYSIZE=<integer>:** Defaults to `256`. Block height in pixels. Maximum supported is 4096.
- **TILE_FORMAT=[PNG_JPEG/PNG/PNG8/JPEG/WEBP/TIFF/AUTO]:** Defaults to `AUTO`. Format used to store tiles. See *Tile formats*.
- **QUALITY=1-100:** Defaults to `75`. Quality setting for JPEG and WEBP compression.
- **ZLEVEL=1-9:** Defaults to `6`. DEFLATE compression level for PNG tiles.

- **DITHER=[YES/NO]:** Defaults to NO. Whether to use Floyd-Steinberg dithering (for *TILE_FORMAT=PNG8*).
- **TILING_SCHEME=[CUSTOM/GoogleCRS84Quad/GoogleMapsCompatible/InspireCRS84Quad/PseudoTMS_GlobalGeodetic/PseudoTMS_GlobalMercator/other]:** Defaults to CUSTOM. See *Tiling schemes*. Starting with GDAL 3.2, the value of TILING_SCHEME can also be the filename of a JSON file according to the *OGC Two Dimensional Tile Matrix Set standard*, a URL to such file, the radical of a definition file in the GDAL data directory (e.g. FOO for a file named tms_FOO.json) or the inline JSON definition. Note: the TILING_SCHEME option with a non-CUSTOM value is best used with the gdal_translate utility / CreateCopy() API operation. If used with gdalwarp, it requires setting the -tr switch to the exact value expected by one zoom level of the tiling scheme.

  In non-CUSTOM mode, the actual resolution can be controlled with the *ZOOM_LEVEL* or *ZOOM_LEVEL_STRATEGY* options.
- **ZOOM_LEVEL=<integer>:** (GDAL >= 3.9) Zoom level number (starting at 0 for coarsest zoom level). Only used for *TILING_SCHEME* different from CUSTOM. If this option is specified, *ZOOM_LEVEL_STRATEGY* is ignored.
- **ZOOM_LEVEL_STRATEGY=[AUTO/LOWER/UPPER]:** Defaults to AUTO. Strategy to determine zoom level. Only used by CreateCopy() for *TILING_SCHEME* different from CUSTOM. LOWER will select the zoom level immediately below the theoretical computed non-integral zoom level, leading to subsampling. On the contrary, UPPER will select the immediately above zoom level, leading to oversampling. Defaults to AUTO which selects the closest zoom level.
- **RESAMPLING=[NEAREST/BILINEAR/CUBIC/CUBICSPLINE/LANCZOS/MODE/AVERAGE]:** Defaults to BILINEAR. Resampling algorithm. Only used by CreateCopy() for TILING_SCHEME different from CUSTOM. Defaults to BILINEAR.
- **PRECISION=<floating_point_value_in_vertical_units>:** Defaults to 1.0. Smallest significant value. Only used for tile gridded coverage datasets.
- **UOM=value:** Unit of Measurement. Only used for tiled gridded coverage datasets. Also set through SetUnitType()
- **FIELD_NAME=value:** Defaults to Height. Field name. Only used for tiled gridded coverage datasets.
- **QUANTITY_DEFINITION=value:** Defaults to Height. Description of the field. Only used for tiled gridded coverage datasets.
- **GRID_CELL_ENCODING=[grid-value-is-center/grid-value-is-area/grid-value-is-corner]:** Grid cell encoding. Only used for tiled gridded coverage datasets. Defaults to grid-value-is-center, when AREA_OR_POINT metadata item is not set.
- **VERSION=[AUTO/1.0/1.1/1.2/1.3/1.4]:** Set GeoPackage version (for application_id and user_version fields). In AUTO mode, this will be equivalent to 1.4 starting with GDAL 3.11 (1.2 in prior versions) 1.3 is available starting with GDAL 3.3 1.4 is available starting with GDAL 3.7.1
- **ADD_GPKG_OGR_CONTENTS=[YES/NO]:** Defaults to YES. Defines whether to add a gpkg_ogr_contents table to keep feature count for vector layers, and associated triggers.
- **CRS_WKT_EXTENSION=[YES/NO]:** (GDAL >= 3.8) Defaults to NO. Defines whether to add the definition_12_063 column to the gpkg_spatial_ref_sys system table, according to http://www.geopackage.org/spec/#extension_crs_wkt . The default is NO, unless the tile gridded coverage extension is used. With VERSION >= 1.4, a epoch column is also added. WKT strings in definition_12_063 will follow the WKT2:2015 standard when possible, but may use the WKT2:2019 standard for specific cases (dynamic CRS with coordinate epoch). This option generally does not need to be specified, as the driver will automatically update the gpkg_spatial_ref_sys table when needed, but it may be useful to create GeoPackage datasets matching the exceptions of other software or profiles (such as the DGIWG-GPKG profile).
- **METADATA_TABLES=[YES/NO]:** (GDAL >= 3.8) Defines whether to add the metadata system tables. By default, they are created on demand. If NO is specified, they are not created, even if metadata is set. If YES is

specified, they are always created.

6.46.4 Overviews

gdaladdo / BuildOverviews() can be used to compute overviews. Power-of-two overview factors (2,4,8,16,...) should be favored to be conformant with the baseline GeoPackage specification. Use of other overview factors will work with the GDAL driver, and cause the gpkg_zoom_other extension to be registered, but that could potentially cause interoperability problems with other implementations that do not support that extension.

Overviews can also be cleared with the -clean option of gdaladdo (or BuildOverviews() with nOverviews=0)

6.46.5 Metadata

GDAL uses the standardized gpkg_metadata and gpkg_metadata_reference tables to read and write metadata.

GDAL metadata, from the default metadata domain and possibly other metadata domains, is serialized in a single XML document, conformant with the format used in GDAL PAM (Persistent Auxiliary Metadata) .aux.xml files, and registered with md_scope=dataset and md_standard_uri=http://gdal.org in gpkg_metadata. In gpkg_metadata_reference, this entry is referenced with a reference_scope=table and table_name={name of the raster table}

It is possible to read and write metadata that applies to the global GeoPackage, and not only to the raster table, by using the *GEOPACKAGE* metadata domain.

Metadata not originating from GDAL can be read by the driver and will be exposed as metadata items with keys of the form GPKG_METADATA_ITEM_XXX and values the content of the *metadata* columns of the gpkg_metadata table. Update of such metadata is not currently supported through GDAL interfaces (although it can be through direct SQL commands).

The specific DESCRIPTION and IDENTIFIER metadata item of the default metadata domain can be used in read/write to read from/update the corresponding columns of the gpkg_contents table.

You can set the CREATE_METADATA_TABLES configuration option to NO to avoid creating and filling the metadata tables.

6.46.6 IMAGE_STRUCTURE metadata item

Note

Implementation details, normally transparent to GDAL users, but useful for other implementations.

Starting with GDAL 3.6.1, the following optional metadata items can be read and write into the `IMAGE_STRUCTURE` metadata domain, in the `<GDALMultiDomainMetadata>` XML element:

- BAND_COUNT=1, 2, 3 or 4. Applies only for Byte data. Set when creating a dataset so that GDAL knows the number of bands when reopening it.
- COLOR_TABLE={{r0,g0,b0,a0},...{r255,g255,b255,a255}}. Applies only for Byte data and a single band dataset. Set when creating a dataset from a source dataset that has a color table.
- TILE_FORMAT=PNG/PNG8/PNG_JPEG/JPEG/WEBP. Set when creating a dataset so that GDAL knows the tile format when reopening it, for updates.
- NODATA_VALUE=integer between 0 and 255. Applies only for Byte data.

Example:

```
INSERT INTO gpkg_metadata VALUES(
    1,
    'dataset',
    'http://gdal.org',
    'text/xml',
    '<GDALMultiDomainMetadata><Metadata domain="IMAGE_STRUCTURE"><MDI key="BAND_COUNT">1
↪</MDI><MDI key="NODATA_VALUE">255</MDI></Metadata></GDALMultiDomainMetadata>')
);
INSERT INTO gpkg_metadata_reference VALUES(
    'table','my_raster_table',NULL,NULL,'2022-11-09T18:44:59.723Z',1,NULL);
```

6.46.7 Level of support of GeoPackage Extensions

(Restricted to those have a raster scope)

Table 3: Extensions

Extension name	OGC adopted extension ?	Supported by GDAL?
Zoom Other intervals	Yes	Yes
Tiles Encoding WebP	Yes	Yes
Metadata	Yes	Yes
WKT for Coordinate Reference Systems (WKT v2)	Yes	Yes
Tiled Gridded Coverage Data	Yes	Yes

6.46.8 Examples

- Simple translation of a GeoTIFF into GeoPackage. The table 'byte' will be created with the tiles.

```
gdal_translate -of GPKG byte.tif byte.gpkg
```

- Translation of a GeoTIFF into GeoPackage using WebP tiles

```
gdal_translate -of GPKG byte.tif byte.gpkg -co TILE_FORMAT=WEBP
```

- Translation of a GeoTIFF into GeoPackage using GoogleMapsCompatible tiling scheme (with reprojection and resampling if needed)

```
gdal_translate -of GPKG byte.tif byte.gpkg -co TILING_SCHEME=GoogleMapsCompatible
```

- Building of overviews of an existing GeoPackage, and forcing JPEG tiles

```
gdaladdo -r cubic -oo TILE_FORMAT=JPEG my.gpkg 2 4 8 16 32 64
```

- Addition of a new subdataset to an existing GeoPackage, and choose a non default name for the raster table.

```
gdal_translate -of GPKG new.tif existing.gpkg -co APPEND_SUBDATASET=YES -co RASTER_
↪TABLE=new_table
```

- Reprojection of an input dataset to GeoPackage

```
gdalwarp -of GPKG in.tif out.gpkg -t_srs EPSG:3857
```

- Open a specific raster table in a GeoPackage

```
gdalinfo my.gpkg -oo TABLE=a_table
```

6.46.8.1 Raster SQL functions

The raster SQL functions mentioned at *Raster related functions* are also available.

The `gdal_get_layer_pixel_value()` function (added in GDAL 3.7), variant of the generic `gdal_get_pixel_value()`, can be used to extract the value of a pixel in a raster layer of the current dataset.

It takes 5 or 6 arguments:

- a string with the layer/table name
- a band number (numbering starting at 1)
- a string being "georef" to indicate that subsequent values will be georeferenced coordinates, or "pixel" to indicate that subsequent values will be in column, line pixel space
- georeferenced X value or column number
- georeferenced Y value or line number
- resampling method among `nearest` (default), `bilinear`, `cubic`, `cubicspline`. Optional, added in GDAL 3.10

```
SELECT gdal_get_layer_pixel_value('my_raster_table', 1, 'georef', 440720, 3751320)
SELECT gdal_get_layer_pixel_value('my_raster_table', 1, 'pixel', 0, 0)
SELECT gdal_get_pixel_value('my_raster_table', 1, 'pixel', 0.5, 0.5, 'bilinear')  --␣
↪GDAL >= 3.10
```

6.46.9 See Also

- *GeoPackage vector* documentation page
- Getting Started With GeoPackage
- OGC GeoPackage format standard specification, HTML format (current/development version of the standard)
- OGC GeoPackage Encoding Standard page
- SQLite
- *PNG driver* documentation page
- *JPEG driver* documentation page
- *WEBP driver* documentation page
- OGC 07-057r7 WMTS 1.0 specification
- OSGeo TMS (Tile Map Service) specification

6.46.10 Other notes

Development of raster support in the GeoPackage driver was financially supported by Safe Software.

6.47 GRASS Raster Format

Driver short name

GRASS

Build dependencies

gdal-grass-driver

The GDAL-GRASS driver can read GRASS GIS raster maps. The driver is maintained at the GDAL-GRASS GIS driver repository.

For the driver documentation, please visit the GDAL-GRASS GIS raster driver page.

6.47.1 See Also

- GRASS GIS home page

6.48 GRASSASCIIGrid -- GRASS ASCII Grid

Driver short name

GRASSASCIIGrid

Driver built-in by default

This driver is built-in by default

Supports reading GRASS ASCII grid format (similar to Arc/Info ASCIIGRID command).

By default, the datatype returned for GRASS ASCII grid datasets by GDAL is autodetected, and set to Float32 for grid with floating point values or Int32 otherwise. This is done by analysing the format of the null value and the first 100k bytes of data of the grid. You can also explicitly specify the datatype by setting the GRASSASCIIGRID_DATATYPE configuration option (Int32, Float32 and Float64 values are supported currently)

NOTE: Implemented as aaigriddataset.cpp.

6.48.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.49 GRIB -- WMO General Regularly-distributed Information in Binary form

Driver short name

GRIB

Driver built-in by default

This driver is built-in by default

GDAL supports GRIB1 (reading) and GRIB2 (reading and writing) format raster data, with support for common coordinate system, georeferencing and other metadata. GRIB format is commonly used for distribution of Meteorological information, and is propagated by the World Meteorological Organization.

The GDAL GRIB driver is based on a modified version of the degrib application which is written primarily by Arthur Taylor of NOAA NWS NDFD (MDL). The degrib application (and the GDAL GRIB driver) are built on the g2clib grib decoding library written primarily by John Huddleston of NOAA NWS NCEP.

GRIB2 files without projection on lon/lat grids have the peculiarity of using longitudes in the [0,360] range and wrapping around the antimeridian as opposed to the usual wrapping around the prime meridian of other raster datasets. Starting with GDAL 3.4.0, when reading such files, a transparent conversion of the longitudes to [-180,180] will be performed and the data will be rewrapped around the prime meridian - the split&swap mode. This behavior can be disabled by setting the *GRIB_ADJUST_LONGITUDE_RANGE* configuration option to *NO*.

There are several encoding schemes for raster data in GRIB format. Most common ones should be supported including PNG encoding. JPEG2000 encoded GRIB files will generally be supported if GDAL is also built with JPEG2000 support via one of the GDAL JPEG2000 drivers.

GRIB files may a be represented in GDAL as having many bands, with some sets of bands representing a time sequence. GRIB bands are represented as Float64 (double precision floating point) regardless of the actual values. GRIB metadata is captured as per-band metadata and used to set band descriptions, similar to this:

```
Description = 100000[Pa] ISBL="Isobaric surface"
  GRIB_UNIT=[gpm]
  GRIB_COMMENT=Geopotential height [gpm]
  GRIB_ELEMENT=HGT
  GRIB_SHORT_NAME=100000-ISBL
  GRIB_REF_TIME=  1201100400 sec UTC
  GRIB_VALID_TIME=  1201104000 sec UTC
  GRIB_FORECAST_SECONDS=3600 sec
```

GRIB2 files may also include an extract of other metadata, such as the identification section, product definition template number (GRIB_PDS_PDTN, octet 8-9), and the product definition template values (GRIB_PDS_TEMPLATE_NUMBERS, octet 10+) as metadata like this:

```
GRIB_DISCIPLINE=0(Meteorological)
GRIB_IDS=CENTER=7(US-NCEP) SUBCENTER=0 MASTER_TABLE=8 LOCAL_TABLE=1 SIGNF_REF_
↪TIME=1(Start_of_Forecast) REF_TIME=2017-10-20T06:00:00Z PROD_STATUS=0(Operational)␣
↪TYPE=1(Forecast)
GRIB_PDS_PDTN=32
GRIB_PDS_TEMPLATE_NUMBERS=5 7 2 0 0 0 0 0 1 0 0 0 0 1 0 31 1 29 67 140 2 0 0 238 217
GRIB_PDS_TEMPLATE_ASSEMBLED_VALUES=5 7 2 0 0 0 0 1 0 1 31 285 17292 2 61145
```

GRIB_DISCIPLINE is the Discipline field of the Section 0 of the message.

GRIB_IDS is the identification section / Section 1 of the message.

GRIB_PDS_TEMPLATE_ASSEMBLED_VALUES use template definitions to assemble several bytes that make a template item into a 16 or 32 bit signed/unsigned integers, whereas GRIB_PDS_TEMPLATE_NUMBERS expose raw bytes

6.49.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.49.2 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). This paragraph lists the configuration options that can be set to alter the default behavior of the GRIB driver.

- **GRIB_ADJUST_LONGITUDE_RANGE=[YES/NO]:** (GDAL >= 3.4.0) Defaults to `YES`. Convert longitudes from [0, 360] to [-180, 180].
- **GRIB_NORMALIZE_UNITS=[YES/NO]:** Defaults to `YES`. Can be set to NO to avoid GDAL to normalize units to metric. By default (*GRIB_NORMALIZE_UNITS=YES*), temperatures are reported in degree Celsius (°C). With *GRIB_NORMALIZE_UNITS=NO*, they are reported in Kelvin (K).
- **GRIB_RESOURCE_DIR=<path>:** Path to a directory where grib2_*.csv tables are located. If not specified, the *GDAL_DATA* configuration option (or hard coded paths) used for all GDAL resources will be used.

6.49.3 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are supported:

- **USE_IDX=[YES/NO]:** (GDAL >= 3.4) Defaults to `YES`. Enable automatic reading of external wgrib2 external index files when available. GDAL will look for a *<GRIB>.idx* in the same place as the dataset. These files when combined with careful usage of the API or the CLI tools allow a GRIBv2 file to be opened without reading all the bands. In particular, this allows an orders of magnitude faster extraction of select bands from large GRIBv2 files on remote storage (like NOMADS on AWS S3). In order to avoid unnecessary I/O only the text description of the bands should be accessed as accessing the metadata will require loading of the band header. gdal_translate is supported but gdalinfo is not. This option is ignored when using the multidimensional API (index is then ignored)

6.49.4 GRIB2 write support

GRIB2 write support is available through the CreateCopy() / gdal_translate interface.

Each band of the input dataset is translated as a GRIB2 message, and all of them are concatenated in a single file, conforming to the usual practice.

The input dataset must be georeferenced, and the supported projections are: Geographic Longitude/Latitude, Mercator 1SP/2SP, Transverse Mercator, Polar Stereographic, Lambert Conformal Conic 1SP/2SP, Albers Conic Equal Area and Lambert Azimuthal Equal Area.

A number of creation options are available as detailed in below sections. Those creation options are valid for all bands. But it is possible to override those global settings in a per-band way, by defining creation options that use the same key and are prefixed by BAND_X_ where X is the band number between 1 and the total number of bands. For example BAND_1_PDS_PDTN

6.49.4.1 Product identification and definition

Users are strongly advised to provide necessary information to appropriately fill the Section 0 / "Indicator", Section 1 / "Identification section" and Section 4 / "Product definition section" with the following creation options. Otherwise, GDAL will fill with default values, but readers might have trouble exploiting GRIB2 datasets generating with those defaults. Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python).

- **DISCIPLINE=<integer>:** sets the Discipline field of Section 0. Valid values are given by Table 0.0:
  - 0: Meteorological Products. Default value
  - 1: Hydrological Products
  - 2: Land Surface Products
  - 3, 4: Space Products
  - 10: Oceanographic Product
- **IDS=value:** String with different elements to fill the fields of the Section 1 / Identification section. The value of that string will typically be retrieved from the GRIB_IDS metadata item of an existing GRIB product. For example "IDS=CENTER=7(US-NCEP) SUBCENTER=0 MASTER_TABLE=8 SIGNF_REF_TIME=1(Start_of_Forecast) REF_TIME=2017-10-20T06:00:00Z PROD_STATUS=0(Operational) TYPE=1(Forecast)". More formally, the format of the string is a list of KEY=VALUE items, with space separator. The accepted keys are CENTER, SUBCENTER, MASTER_TABLE, SIGNF_REF_TIME, REF_TIME, PROD_STATUS and TYPE. Only the numerical part of the value is taken into account (the precision between parenthesis will be ignored). It is possible to use both this IDS creation option and a specific IDS_xxx creation option that will override the potential corresponding xxx key of IDS. For example with the previous example, if both "IDS=CENTER=7(US-NCEP)..." and "IDS_CENTER=8" are define, the actual value used with be 8.
- **IDS_CENTER=<integer>:** Defaults to `255/Missing`. Identification of originating/generating center, according to Table 0.
- **IDS_SUBCENTER=<integer>:** Defaults to `65535/Missing`. Identification of originating/generating center, according to Table C.
- **IDS_MASTER_TABLE=<integer>:** Defaults to `2`. GRIB master tables version number, according to Table 1.0.
- **IDS_SIGNF_REF_TIME=<integer>:** Defaults to `0/Analysis`. Significance of reference time, according to Table 1.2.
- **IDS_REF_TIME=<YYYY-MM-DD[THH:MM:SSZ]>:** Defaults to `1970-01-01T00:00:00Z`. Reference time.

- **IDS_PROD_STATUS=<integer>:** Defaults to `255/Missing`. Production status of processed data, according to Table 1.3.
- **IDS_TYPE=<integer>:** Defaults to `255/Missing`. Type of processed data, according to Table 1.4.
- **PDS_PDTN=<integer>:** Product definition template number, according to Table 4.0. Defaults to 0/Analysis or forecast at a horizontal level or in a horizontal layer at a point in time. If this default template number is used, and none of PDS_TEMPLATE_NUMBERS or PDS_TEMPLATE_ASSEMBLED_VALUES is specified, then a default template definition is also used, with most fields set to Missing.
- **PDS_TEMPLATE_NUMBERS=value:** Product definition template raw numbers. This is a list of byte values (between 0 and 255 each), space separated. The number of values and their semantics depends on the template number specified by PDS_PDTN, and you have to consult the template structures pointed by Table 4.0. It might be easier to use the GRIB_PDS_TEMPLATE_NUMBERS reported by existing GRIB2 products as the value for this item. If the template structure is known by the reading side of the driver, an effort to validate the number of template numbers against the template structure is made (with warnings if more elements than needed are specified, and error if less are specified). It is also possible to define a template that is not or partially implemented by the reading side of the driver.
- **PDS_TEMPLATE_ASSEMBLED_VALUES=value:** Product definition template assembled values. This is a list of values (with the range of signed/unsigned 1, 2 or 4-byte wide integers, depending on the item), space separated. The number of values and their semantics depends on the template number specified by PDS_PDTN, and you have to consult the template structures pointed by Table 4.0. It might be easier to use the GRIB_PDS_TEMPLATE_ASSEMBLED_VALUES reported by existing GRIB2 products as the value for this item. PDS_TEMPLATE_NUMBERS and PDS_TEMPLATE_ASSEMBLED_VALUES are exclusive. To use this creation option, the template structure must be known by the reading side of the driver.

6.49.4.2 Data encoding

In GRIB2, a number of data encoding schemes exist (see Section 5 / "Data representation section"). By default, GDAL will select an appropriate data encoding that will preserve the range of input data. with the *DATA_ENCODING*, *NBITS*, *DECIMAL_SCALE_FACTOR*, *JPEG2000_DRIVER*, *COMPRESSION_RATIO* and *SPATIAL_DIFFERENCING_ORDER* creation options.

Users can override those defaults with the following creation options are:

- **DATA_ENCODING=[AUTO/SIMPLE_PACKING/COMPLEX_PACKING/IEEE_FLOATING_POINT/PNG/JPEG2000]:** Defaults to `AUTO`. Choice of the Data representation template number.
  - In AUTO mode, COMPLEX_PACKING is selected if input band has a nodata value. Otherwise if input band datatype is Float32 or Float64, IEEE_FLOATING_POINT is selected. Otherwise SIMPLE_PACKING is selected.
  - SIMPLE_PACKING: use integer representation internally, with offset and decimal and/or binary scaling. So can be used for any datatype.
  - COMPLEX_PACKING: evolution of SIMPLE_PACKING with nodata handling. By default, a non-spatial differencing encoding is used, but if SPATIAL_DIFFERENCING_ORDER=1 or 2, complex packing with spatial differencing is used
  - IEEE_FLOATING_POINT: store values as IEEE-754 single or double precision numbers.
  - PNG: uses the same preparation steps as SIMPLE_PACKING but with PNG encoding of the integer values.
  - JPEG2000: uses the same preparation steps as SIMPLE_PACKING but with JPEG2000 encoding of the integer values.
- **NBITS=1-31:** Bit width for each sample value. Might be only loosely honored by some *DATA_ENCODING*. If not specified, the bit width is computed automatically from the range of input values for integral data types, or default to 8 for Float32/Float64.

- **DECIMAL_SCALE_FACTOR=<integer>:** Defaults to 0. Input values are multiplied by 10^DECIMAL_SCALE_FACTOR before integer encoding (and automatically divised by this value at decoding, so this only affect precision). For example, if the type of the data is a temperature, with floating point data type, DECIMAL_SCALE_FACTOR=1 can be used to specify that the data has a precision of 1/10 of degree. The default is 0 (no premultiplication)
- **SPATIAL_DIFFERENCING_ORDER=[0/1/2]:** Defaults to 0. Only used for *DATA_ENCODING=COMPLEX_PACKING*. Defines the order of the spatial differencing. 0 means that the values are encoded independently, 1 means that the difference of consecutive values is encoded and 2 means that the difference of the difference of consecutive values is encoded.
- **COMPRESSION_RATIO=1-100:** Defaults to 1. Defaults to 1 for lossless JPEG2000 encoding. Only used for JPEG2000 encoding. If a value greater than 1 is specified, lossy JPEG2000 compression is used. The value indicates the desired compression factor with respected to uncompressed data. For example a value of 10 means that the desired JPEG2000 codestream should be 10 times smaller than the corresponding uncompressed file (with NBITS bits per pixel).
- **JPEG2000_DRIVER=[JP2KAK/JP2OPENJPEG/JPEG2000/JP2ECW]:** (possible values depend on the actually available JPEG2000 driver in the GDAL build). To specify which JPEG2000 driver should be used. If not specified, drivers are searched in the order given in the enumeration.

6.49.4.3 Data units

Internally GRIB stores values in the units of the international system (ie Metric system). So temperatures must be stored as Kelvin degrees. But on the reading side of the driver, fields with temperatures are exposed in Celsius degrees (unless the *GRIB_NORMALIZE_UNITS* configuration option is set to NO). For consistency, the writing side of the driver also assumed that temperature (detected if the first value of a product definition template, ie the *Parameter category* is 0=Temperature) values in the input dataset will be in Celsius degrees, and will automatically offset them to Kelvin degrees. It is possible to control that behavior using the *INPUT_UNIT* creation option:

- **INPUT_UNIT=[C/K]:** Defaults to C. Used to specify temperature units when writing: C (for Celsius) or K (for Kelvin).

6.49.4.4 GRIB2 to GRIB2 conversions

If GRIB2 to GRIB2 translation is done with gdal_translate (or CreateCopy()), the GRIB_DISCIPLINE, GRIB_IDS, GRIB_PDS_PDTN and GRIB_PDS_TEMPLATE_NUMBERS metadata items of the bands of the source dataset are used by default (unless creation options override them).

DECIMAL_SCALE_FACTOR and NBITS will also be attempted to be retrieved from the GRIB special metadata domain.

6.49.4.5 Examples

```
gdal_translate in.tif out.grb2 -of GRIB \
    -co "IDS=CENTER=8(US-NWSTG) SIGNF_REF_TIME=1(Start_of_Forecast) REF_TIME=2008-02-
↪21T17:00:00Z PROD_STATUS=0(Operational) TYPE=1(Forecast)" \
    -co "PDS_PDTN=8" \
    -co "PDS_TEMPLATE_ASSEMBLED_VALUES=0 5 2 0 0 255 255 1 43 1 0 0 255 -1 -2147483647␣
↪2008 2 23 12 0 0 1 0 3 255 1 12 1 0"
```

6.49.5 See Also:

- NOAA NWS NDFD "degrib" GRIB2 Decoder
- NOAA NWS NCEP g2clib grib decoding library
- WMO GRIB1 Format Documents

- NCEP WMO GRIB2 Documentation

6.49.6 Credits

Support for GRIB2 write capabilities has been funded by Meteorological Service of Canada.

6.50 GS7BG -- Golden Software Surfer 7 Binary Grid File Format

Driver short name

GS7BG

Driver built-in by default

This driver is built-in by default

This is the binary (non-human-readable) version of one of the raster formats used by Golden Software products (such as the Surfer series). This format differs from the *GSBG -- Golden Software Binary Grid File Format* format (also known as Surfer 6 binary grid format), it is more complicated and flexible.

NOTE: Implemented as gs7bgdataset.cpp.

6.50.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.51 GSAG -- Golden Software ASCII Grid File Format

Driver short name

GSAG

Driver built-in by default

This driver is built-in by default

This is the ASCII-based (human-readable) version of one of the raster formats used by Golden Software products (such as the Surfer series). This format is supported for both reading and writing (copy mode only). Currently the associated formats for color, metadata, and shapes are not supported.

NOTE: Implemented as gsagdataset.cpp.

6.51.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.52 GSBG -- Golden Software Binary Grid File Format

Driver short name

GSBG

Driver built-in by default

This driver is built-in by default

This is the binary (non-human-readable) version of one of the raster formats used by Golden Software products (such as the Surfer series).

Like the ASCII version, this format is supported for both reading and writing (including create, delete, and copy). Currently the associated formats for color, metadata, and shapes are not supported.

Surfer 8 uses a fixed nodata value at 1.701410009187828e+38. When writing a GSBG dataset, GDAL remaps the user-specified input nodata value to 1.701410009187828e+38.

NOTE: Implemented as gsbgdataset.cpp.

6.52.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.53 GSC -- GSC Geogrid

Driver short name

GSC

Driver built-in by default

This driver is built-in by default

NOTE: Implemented as gscdataset.cpp.

6.53.1 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.54 GDALG: GDAL Streamed Algorithm

Added in version 3.11.

Driver short name

GDALG

Driver built-in by default

This driver is built-in by default

Note

GDALG is the contraction of GDAL and ALGorithm.

This is a read-only driver that reads a JSON file containing an invocation of the *gdal command line interface*, that results in a on-the-fly / streamed raster dataset when used as an input to *GDALOpenEx()*, or anywhere else in GDAL where a raster input dataset is expected. GDALG files are conceptually close to *VRT (Virtual) files*, although the implementation is substantially different.

The subset of commands of **gdal** that can generate such streamed datasets indicate support for writing `GDALG` in the `--help` description for `--output-format`. This includes *gdal raster pipeline*, *gdal raster mosaic* or *gdal raster stack*.

It is recommended that GDALG files use the `.gdalg.json` extension.

The JSON document must include a `type` member with the value `gdal_streamed_alg`, and a `command_line` member, which is very close to the one that would be used to generate a materialized dataset, but using the `stream` output format.

```
{
    "type": "gdal_streamed_alg",
    "command_line": "gdal raster mosaic in1.tif in2.tif --output-format stream streamed_
↪dataset"
}
```

If such file is saved as `mosaic.gdalg.json`, it can be for example read with `gdal raster info mosaic.gdalg.json`.

In the case of a *gdal raster pipeline*, the final `write` step can be omitted.

```
{
    "type": "gdal_streamed_alg",
    "command_line": "gdal raster pipeline ! read in.tif ! reproject --dst-crs=EPSG:32632"
}
```

An optional `relative_paths_relative_to_this_file` boolean member defaults to `true`, to indicate that relative input filenames in the command line should be interpreted as relative to the location of the `.gdalg.json` file. If setting this member to `false`, they will be interpreted as being relative to the current working directory.

`.gdalg.json` files can be validated against the following JSON schema gdalg.schema.json

This driver has also vector capabilities as detailed in *GDALG: GDAL Streamed Algorithm*.

6.54.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

6.55 GTA - Generic Tagged Arrays

Driver short name

GTA

Build dependencies

libgta

GDAL can read and write GTA data files through the libgta library.

GTA is a file format that can store any kind of multidimensional array data, allows generic manipulations of array data, and allows easy conversion to and from other file formats.

6.55.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Georeferencing

This driver supports georeferencing

6.55.2 Creation options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following creation options are supported:

- **COMPRESS=[NONE/BZIP2/XZ/ZLIB/ZLIB1/ZLIB2/ZLIB3/ZLIB4/ZLIB5/ZLIB6/ZLIB7/ZLIB8/ZLIB9]:** Defaults to `NONE`. Set the GTA compression method.

6.55.3 See Also

- GTA home page

6.56 GTI -- GDAL Raster Tile Index

Added in version 3.9.

Driver short name

GTI

Driver built-in by default

This driver is built-in by default

6.56.1 Introduction

The GTI driver is a driver that allows to handle catalogs with a large number of raster files (called "tiles" in the rest of this document, even if a regular tiling is not required by the driver), and build a virtual mosaic from them. Each tile may be in any GDAL supported raster format, and be a file stored on a regular filesystem, or any GDAL supported virtual filesystem (for raster drivers that support such files).

This driver offers similar functionality as the *VRT* driver with the following main differences:

- The tiles are listed as features of any GDAL supported vector format. Use of formats with efficient spatial filtering is recommended, such as *GeoPackage*, *FlatGeoBuf* or *PostGIS*. The GTI driver can thus use a larger number of tiles than the VRT driver (hundreds of thousands or more), provided the underlying vector format is efficient.
- The tiles may have different SRS. The GTI driver is capable of on-the-fly reprojection
- The GTI driver offers control on the order in which tiles are composited, when they overlap (z-order)
- The GTI driver honours the mask/alpha band when compositing together overlapping tiles.
- Contrary to the VRT driver, the GTI driver does not enable to alter characteristics of referenced tiles, such as their georeferencing, nodata value, etc. If such behavior is desired, the tiles must be for example wrapped individually in a VRT file (or *vrt://* connection string) before being referenced in the GTI index.

6.56.2 Connection strings

The GTI driver accepts different types of connection strings:

- a vector file in GeoPackage format with a `.gti.gpkg` extension, or in FlatGeoBuf format with a `.gti.fgb` extension, meeting the minimum requirements for a GTI compatible tile index, detailed later.

  For example: `tileindex.gti.gpkg`

  Starting with GDAL 3.10, specifying the `-if GTI` option to command line utilities accepting it, or `GTI` as the only value of the `papszAllowedDrivers` of *GDALOpenEx()*, also forces the driver to recognize the passed filename if its extension is just `.gpkg` or `.fgb`.

- any vector file in a GDAL supported format, with its filename (or connection string prefixed with `GTI:`

  For example: `GTI:tileindex.shp` or `GTI:PG:database=my_db schema=tileindex`

- a XML file, following the below GTI XML format, generally with the recommended `.gti` extension, referencing a vector file. Using such XML file may be more practical for tile indexes not stored in a file, or if some additional metadata must be defined at the dataset or band level of the virtual mosaic.

  For example: `tileindex.gti`

6.56.3 STAC GeoParquet support

Added in version 3.10.

The driver can support STAC GeoParquet catalogs, provided GDAL is built with *(Geo)Parquet* support. It can make use of fields (`proj:code`, `proj:epsg`, `proj:wkt2`, or `proj:projson`) and `proj:transform` from the Projection Extension Specification, to correctly infer the appropriate projection and resolution.

Since GDAL 3.13, program *gdal raster index* can generate such catalogs with the `--profile STAC-GeoParquet` option.

Example of a valid connection string: `GTI:/vsicurl/https://github.com/stac-utils/stac-geoparquet/raw/main/tests/data/naip.parquet`

6.56.4 Tile index requirements

The minimum requirements for a GTI compatible tile index is to be a vector format supported by GDAL, with a geometry column storing polygons with the extent of the tiles, and an attribute field of type string, storing the path to each tile. The default name for this attribute field is `location`. If relative filenames are stored in the tile index, they are considered to be relative to the path of the tile index.

In addition, for formats that can store layer metadata (GeoPackage, FlatGeoBuf, PostGIS, ...), the following layer metadata items may be set:

- `RESX=<float>` and `RESY=<float>`: resolution along X and Y axis, in SRS units / pixel.

  Setting those metadata items is recommended, otherwise the driver will try to open one of the tiles referenced in the tile index, and use its resolution as the resolution for the mosaic.

- `BAND_COUNT=<int>`: number of bands of the virtual mosaic. The tiles stored in an index should generally have the same number of bands.

  Setting that metadata item is recommended, otherwise the driver will try to open one of the tiles referenced in the tile index, and use it as the number of bands for the mosaic.

  A mix of tiles with N and N+1 bands is allowed, provided that the color interpretation of the (N+1)th band is alpha. The N+1 value must be written as the band count in that situation.

  If tiles contains a single band with a color table, and that the color table may differ among tiles, BAND_COUNT should be set to 3 (resp. 4) to perform expansion to Red, Green, Blue components (resp. Red, Green, Blue, Alpha). If the color table is identical between the tiles, and it is desired to preserve it, the VRRTI XML file format may be used to include the ColorTable element.

- `DATA_TYPE=<val>`: data type of the tiles of the tile index `Byte`, `Int8`, `UInt16`, `Int16`, `UInt32`, `Int32`, `UInt64`, `Int64`, `Float32`, `Float64`, `CInt16`, `CInt32`, `CFloat32` or `CFloat64`

  Setting that metadata item is recommended, otherwise the driver will try to open one of the tiles referenced in the tile index, and use it as the data type for the mosaic.

- `NODATA=<val>[,<val]...`: nodata value of the bands of the virtual mosaic.

  Note that source tiles may have or may not have a nodata value themselves, and it may be different than the nodata value of the virtual mosaic.

- `MINX=<float>`, `MINY=<float>`, `MAXX=<float>` and `MAXY=<float>`: defines the extent of the virtual mosaic.

  For vector formats that have efficient retrieval of the layer extent, setting those items is not needed.

- `GEOTRANSFORM=<gt0>,<gt1>,<gt2>,<gt3>,<gt4>,<gt5>`: defines the GeoTransform. Used together with `XSIZE` and `YSIZE`, this is an alternate way of defining the extent and resolution os the virtual mosaic.

  It is not necessary to define this item if `RESX=` and `RESY` are set (potentially accompanied with `MINX`, `MINY`, `MAXX` and `MAXY`)

- XSIZE=<int>, YSIZE=<int>: size of the virtual mosaic in pixel.
- COLOR_INTERPRETATION=<val>[,<val]...: color interpretation of the bands of the mosaic. Possible values are red, green, blue, alpha, undefined
- SRS=<string>: specifies the SRS of the virtual mosaic, using any value supported by the *OGRSpatialReference::SetFromUserInput()* call, which includes EPSG Projected, Geographic or Compound CRS (i.e. EPSG:4296), a well known text (WKT) CRS definition, PROJ.4 declarations, etc.

  It is not necessary to define this element if the virtual mosaic SRS is recorded as the SRS of the vector layer of the tile index.

  When SRS is set and the vector layer already has a different SRS, a warning is emitted unless SRS_BEHAVIOR is also set. See SRS_BEHAVIOR below.
- SRS_BEHAVIOR=OVERRIDE|REPROJECT: controls how the SRS parameter interacts with an existing SRS on the vector layer.
  - OVERRIDE: use the specified SRS as the dataset SRS without reprojecting tile coordinates. Use this when the layer's SRS metadata is missing or incorrect and you want to correct it.
  - REPROJECT: reproject tile coordinates from the layer's native SRS to the specified SRS on the fly, including transforming spatial filters applied during raster reads. Use this when you want the virtual mosaic in a different projection than the tile index layer.

  If SRS is set and differs from the vector layer's SRS and SRS_BEHAVIOR is not specified, a warning is emitted. If the vector layer has no SRS, the specified SRS is used silently regardless of this option.

  > **Note**
  >
  > For on-the-fly reprojection of the whole GTI mosaic output, wrapping the GTI file in a warped VRT (e.g. via gdal raster reproject) is an alternative that does not require SRS_BEHAVIOR.

- LOCATION_FIELD=<string>: name of the field where the tile location is stored. Defaults to location.
- SORT_FIELD=<string>: name of a field to use to control the order in which tiles are composited, when they overlap (z-order). That field may be of type String, Integer, Integer64, Date or DateTime. By default, the higher the value in that field, the last the corresponding tile will be rendered in the virtual mosaic (unless SORT_FIELD_ASC=NO is set)
- SORT_FIELD_ASC=YES|NO: whether the values in SORT_FIELD should be sorted in ascending or descending order. Defaults to YES (ascending)
- BLOCKXSIZE=<int> and BLOCKYSIZE=<int>: Block size of bands of the virtual mosaic. Defaults to 256x256.
- MASK_BAND=YES|NO: whether a dataset-level mask band should be exposed. Defaults to NO.
- RESAMPLING=<val>: Resampling method to use for on-the-fly reprojection, or rendering of tiles whose origin coordinates are not at an offset multiple of the resolution of the mosaic compared to the origin of the mosaic. In that later case, RESAMPLING is only taken into account when requesting pixels with the default nearest resampling mode.

  Possible values: nearest, cubic, cubicspline, lanczos, average, rms, gauss

  Defaults to nearest
- BAND_<number>_OFFSET=<val> where number is an integer index starting at 1.

  Additive offset to apply to the raw numbers of the band.
- BAND_<number>_SCALE=<val> where number is an integer index starting at 1.

Multiplicative factor to apply to the raw numbers of the band.

- BAND_<number>_UNITTYPE=<val> where number is an integer index starting at 1.

  Unit of the band.

- OVERVIEW_<idx>_DATASET=<string>. See *Overviews as metadata items of a GDAL vector layer*
- OVERVIEW_<idx>_LAYER=<string>. See *Overviews as metadata items of a GDAL vector layer*
- OVERVIEW_<idx>_OPEN_OPTIONS=<key1=value1>[,key2=value2]... . See *Overviews as metadata items of a GDAL vector layer*
- OVERVIEW_<idx>_FACTOR=<int>. See *Overviews as metadata items of a GDAL vector layer*
- INTERLEAVE=<val> (starting with GDAL 3.13) where <val> can be PIXEL or BAND specifies how pixels belonging to multiple bands are grouped together. This setting affects performance, both how external code accesses pixel values, and internal operation of the driver. When using *gdal driver gti create*, the value is automatically set to match interleave mode among the input sources. Accessing sources whose interleave mode differs from the exposed value may result in reduced performance. See *Multiband pixel organization (INTERLEAVE metadata item)* for more details.

All overviews *must* have exactly the same extent as the full resolution virtual mosaic. The GTI driver does not check that, and if that condition is not met, subsampled pixel request will lead to incorrect result.

They also must be listed by decreasing size with increasing overview index.

In addition to those layer metadata items, the dataset-level metadata item TILE_INDEX_LAYER may be set to indicate, for dataset with multiple layers, which one should be used as the tile index layer.

Since GDAL 3.12, the dataset-level metadata item TILE_INDEX_SQL can be used to specify a SQL request that returns a result set, in addition to or in replacement to TILE_INDEX_LAYER. The dataset-level metadata item TILE_INDEX_SPATIAL_SQL can be used as a complement to TILE_INDEX_SQL for a SQL request that contains placeholders {XMIN}, {YMIN}, {XMAX}, {YMAX} that are substituted at runtime with the coordinates of the requested area of interest.

For example, given a tileindex table with fields location, tile_id and version, where several features may have the same tile_id value, but with different version, the following SQL request (to be used for example in a Spatialite or GeoPackage database) can be use to always select tiles with the latest version:

```
WITH target_version AS (SELECT tile_id,max(version) AS version FROM tileindex GROUP BY␣
↪tile_id)
SELECT * FROM tileindex INNER JOIN target_version
ON target_version.tile_id=tileindex.tile_id AND target_version.version=tileindex.version
```

And in a Spatialite database, the following can be used as the TILE_INDEX_SPATIAL_SQL:

```
WITH target_version AS (SELECT tile_id,max(version) AS version FROM tileindex GROUP BY␣
↪tile_id)
SELECT * FROM tileindex INNER JOIN target_version
ON target_version.tile_id=tileindex.tile_id AND target_version.version=tileindex.version
INNER JOIN idx_tileindex_geometry ON tileindex.ogc_fid = idx_tileindex_geometry.pkid
WHERE idx_tileindex_geometry.xmin <= {XMAX} and idx_tileindex_geometry.ymin <= {YMAX} AND
      idx_tileindex_geometry.xmax >= {XMIN} and idx_tileindex_geometry.ymax >= {YMIN}
```

Alternatively to setting those metadata items individually, the corresponding information can be grouped together in a GTI XML document, attached in the xml:GTI metadata domain of the layer (for drivers that support alternate metadata domains such as GeoPackage)

6.56.5 GTI XML format

A XML schema of the GDAL GTI format (gdaltileindex.xsd) is available.

The following artificial example contains all potential elements and attributes. A number of them have similar name and same semantics as layer metadata items mentioned in the previous section.

```
<GDALTileIndexDataset>
    <IndexDataset>PG:dbname=my_db</IndexDataset>   <!-- required for standalone XML GTI␣
↪files. Ignored if embedded in the xml:GTI metadata domain of the layer  -->
    <IndexLayer>my_layer</IndexLayer>              <!-- optional, but required if there␣
↪are multiple layers in IndexDataset -->
    <SQL>SQL statement</SQL>                       <!-- optional (since 3.12).␣
↪Complement or replacement to IndexLayer -->
    <SpatialSQL>SQL statement</SpatialSQL>         <!-- optional (since 3.12).␣
↪Complement to SQL -->
    <Filter>pub_date >= '2023/12/01'</Filter>      <!-- optional -->
    <SortField>pub_date</SortField>                <!-- optional -->
    <SortFieldAsc>true</SortFieldAsc>              <!-- optional -->
    <SRS>EPSG:4326</SRS>                           <!-- optional -->
    <ResX>60</ResX>                                <!-- optional, but recommended -->
    <ResY>60</ResY>                                <!-- optional, but recommended -->
    <MinX>0</MinX>                                 <!-- optional -->
    <MinY>1</MinY>                                 <!-- optional -->
    <MaxX>2</MaxX>                                 <!-- optional -->
    <MaxY>3</MaxY>                                 <!-- optional -->
    <GeoTransform>2,1,0,49,0,-1</GeoTransform>     <!-- optional -->
    <XSize>2048</XSize>                            <!-- optional -->
    <YSize>1024</YSize>                            <!-- optional -->
    <BlockXSize>256</BlockXSize>                   <!-- optional -->
    <BlockYSize>256</BlockYSize>                   <!-- optional -->
    <Resampling>Cubic</Resampling>                 <!-- optional -->
    <BandCount>1</BandCount>                       <!-- optional, not needed if Band␣
↪elements are defined -->

    <!-- Band is optional, but recommended. Repeated as many times as there are bands -->
    <!-- The "band" attribute is required -->
    <!-- The "dataType" attribute is optional, but recommended -->
    <Band band="1" dataType="Byte">
        <Description>my band</Description>         <!-- optional -->
        <Offset>2</Offset>                         <!-- optional -->
        <Scale>3</Scale>                           <!-- optional -->
        <NoDataValue>4</NoDataValue>               <!-- optional -->
        <UnitType>dn</UnitType>                    <!-- optional -->
        <ColorInterp>Gray</ColorInterp>            <!-- optional -->
        <ColorTable>                               <!-- optional -->
            <Entry c1="1" c2="2" c3="3" c4="255"/>
        </ColorTable>
        <CategoryNames>                            <!-- optional -->
            <Category>cat</Category>
        </CategoryNames>
        <GDALRasterAttributeTable><!--... --></GDALRasterAttributeTable>  <!-- optional -
↪->
        <Metadata>                                 <!-- optional -->
```

(continues on next page)

(continued from previous page)

```
            <MDI key="FOO">BAR</MDI>
        </Metadata>
        <Metadata domain="other_domain">           <!-- optional -->
            <MDI key="FOO">BAR</MDI>
        </Metadata>
    </Band>

    <Metadata>                                     <!-- optional -->
        <MDI key="FOO">BAR</MDI>
    </Metadata>
    <Metadata domain="other_domain">               <!-- optional -->
        <MDI key="FOO">BAR</MDI>
    </Metadata>

    <!-- Overview specification.
         Not required, but overviews must be explicitly specified if desired.
    -->
    <Overview>
        <!-- 1st overview level will reuse the tile index of the
             IndexDataset and IndexLayer elements, with all tiles considered
             downsampled by a factor of 2 -->
        <Factor>2</Factor>
    </Overview>
    <Overview>                                     <!-- optional -->
        <!-- 2nd overview level will reuse the tile index of the
             IndexDataset and IndexLayer elements, with all tiles considered
             downsampled by a factor of 4 -->
        <Factor>4</Factor>
    </Overview>
    <Overview>                                     <!-- optional -->
        <!-- 3rd overview level using a GeoTIFF file. If that file has itself
             overviews, they will also be added.
             To only specify the full resolution of the designated dataset,
             explicitly add <Factor>1</Factor>
        -->
        <Dataset>some.tif</Dataset>
    </Overview>
    <Overview>                                     <!-- optional -->
        <!-- 4th overview level. -->
        <Dataset>another.tif</Dataset>
        <! -- The reduction factor is relative to the size of the full
              resolution layer of the designated dataset,
              and *not* relative to the size of the full
              resolution layer of the GTI dataset.
              It is also possible to select a particular overview by
              using the OVERVIEW_LEVEL open option.
        -->
        <Factor>4</Factor>
    </Overview>
    <Overview>                                     <!-- optional -->
        <!-- 5th overview level. -->
        <Dataset>another.tif</Dataset>
```

(continues on next page)

(continued from previous page)

```
        <! -- Use the overview level of index 2 -->
         <OpenOptions>
              <OOI key="OVERVIEW_LEVEL">2</OOI>
         </OpenOptions>
    </Overview>
    <Overview>                                      <!-- optional -->
        <!-- Last overview level points to another GTI dataset -->
        <Dataset>other.gti.gpkg</Dataset>
        <Layer>other_layer</Layer>                  <!-- optional -->
        <OpenOptions>                               <!-- optional -->
            <OOI key="MINX">2</OOI>
            <OOI key="MINY">49</OOI>
            <OOI key="MAXX">3</OOI>
            <OOI key="MAXY">50</OOI>
        </OpenOptions>
    </Overview>

    <Interleave>Band|Pixel</Interleave>             <!-- optional -->

</GDALTileIndexDataset>
```

At the GDALTileIndexDataset level, the elements specific to GTI XML are:

- `Filter`: value of a SQL WHERE clause, used to select a subset of the features of the index.
- `BlockXSize` / `BlockYSize`: dimension of the block size of bands. Defaults to 256x256
- `Metadata`: defines dataset-level metadata. You can refer to the documentation of the *VRT* driver for its syntax.

At the Band level, the elements specific to GTI XML are: Description, Offset, Scale, UnitType, ColorTable, CategoryNames, GDALRasterAttributeTable, Metadata. You can refer to the documentation of the *VRT* driver for their syntax and semantics.

6.56.6 How to build a GTI compatible index ?

The *gdaltindex* program, or starting with GDAL 3.11, *gdal driver gti create*, may be used to generate both a vector tile index, and optionally a wrapping .gti XML file.

A GTI compatible index may also be created by any programmatic means, provided the above format specifications are met.

6.56.7 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are available. Most of them can be also defined as layer metadata items or in the .gti XML file

- **LAYER=<string>:** For dataset with multiple layers, indicates which one should be used as the tile index layer. Same role as the TILE_INDEX_LAYER dataset level metadata item
- **SQL=<string>:** (GDAL >= 3.12) SQL request that returns a result set, in addition to or in replacement to LAYER. Same role as the TILE_INDEX_SQL dataset level metadata item.
- **SPATIAL_SQL=<string>:** (GDAL >= 3.12) SQL request that contains placeholders `{XMIN}`, `{YMIN}`, `{XMAX}`, `{YMAX}` that are substituted at runtime with the coordinates of the requested area of interest. Complement to *`SQL`*. Same role as the TILE_INDEX_SPATIAL_SQL dataset level metadata item.
- **LOCATION_FIELD=<string>:** Defaults to `location`. Name of the field where the tile location is stored.

- **SORT_FIELD=<string>:** Name of a field to use to control the order in which tiles are composited, when they overlap (z-order). That field may be of type String, Integer, Integer64, Date or DateTime. By default, the higher the value in that field, the last the corresponding tile will be rendered in the virtual mosaic (unless SORT_FIELD_ASC=NO is set)
- **SORT_FIELD_ASC=[YES/NO]:** Defaults to `YES`. Whether the values in SORT_FIELD should be sorted in ascending or descending order
- **FILTER=<string>:** Value of a SQL WHERE clause, used to select a subset of the features of the index.
- **RESX=<float>:** Resolution along X axis in SRS units / pixel.
- **RESY=<float>:** Resolution along Y axis in SRS units / pixel.
- **SRS=<string>:** Specifies the Spatial Reference System in one of the formats supported by *`OGRSpatialReference::SetFromUserInput()`*. When set and the vector layer already has a different SRS, see `SRS_BEHAVIOR`.
- **SRS_BEHAVIOR=[OVERRIDE/REPROJECT]:** Controls how `SRS` interacts with an existing SRS on the vector layer. `OVERRIDE` sets the dataset SRS without reprojecting tile coordinates. `REPROJECT` reprojects tile coordinates from the layer's native SRS to the specified SRS on the fly. If `SRS` is set and differs from the layer's SRS and this option is not specified, a warning is emitted.
- **MINX=<float>:** Minimum X value for the virtual mosaic extent
- **MINY=<float>:** Minimum Y value for the virtual mosaic extent
- **MAXX=<float>:** Maximum X value for the virtual mosaic extent
- **MAXY=<float>:** Maximum Y value for the virtual mosaic extent
- **WARPING_MEMORY_SIZE.=<string>:** (GDAL >= 3.12.3) Defaults to `64MB`. Set the amount of memory that the warp API is allowed to use for caching when on-the-fly reprojection occurs. The value can be specified either as a fixed amount of memory (e.g. `200MB`, `1G`) or as a percentage of usable RAM (`10%`). Note that, in case of multi-threaded optimizations described in the paragraph below, the value applies for each warped source.

6.56.8 Overviews

By default the minimal configuration of a GTI dataset does not expose explicit overviews. But downsampled pixel requests to a GTI dataset may use overviews of the source tiles referenced by the GTI dataset when they exist.

It is possible to provide explicit overview levels that the GTI dataset will expose to external users (and of course use for downsampled pixel requests). This can be done through one or several `<Overview>` elements in the GTI XML format, or by using specific layer metadata keys starting with `OVERVIEW_{idx}` when using a vector dataset as the GTI dataset.

In both cases, each added overview level may be: - a downsampled version of the main GTI dataset - another GTI dataset, at its full resolution or at a subsampled one - any other GDAL recognized raster file, at its full resolution or at a subsampled one.

The order of declaration may matter. Overview levels directly or indirectly specified by later XML or metadata items are only added if their size is smaller than the previously added overview.

6.56.8.1 Overviews in XML

The general syntax for each overview level is:

```
<Overview>
    <Dataset>other.gti.gpkg</Dataset>
    <Layer>other_layer</Layer>
    <Factor>numeric_value_larger_than_one>
    <OpenOptions>
```

(continues on next page)

(continued from previous page)

```
        <OOI key="key">value</OOI>
    </OpenOptions>
</Overview>
```

All elements are optional, but at least one of `Dataset`, `Layer` or `Factor` must be specific. Note also that at most one of those child elements is allowed per `<Overview>` element. So typically if several levels from a same dataset are needed, one `<Overview>` element per level must be specified.

Example 1

This example uses the tile index of the IndexDataset and IndexLayer elements, with all source tiles downsampled by a factor of 2.

```
<Overview>
    <Factor>2</Factor>
</Overview>
```

Example 2

This example uses a GeoTIFF file, and all its potential overviews, as overviews of the GTI dataset

```
<Overview>
    <Dataset>some.tif</Dataset>
</Overview>
```

Example 3

This example uses the overview level of a GeoTIFF file whose dimensions are at least four times smaller than the full resolution layer of `another.tif`.

```
<Overview>
    <Dataset>another.tif</Dataset>
    <Factor>4</Factor>
</Overview>
```

Example 4

This example uses the overview of index 2 of a GeoTIFF file (so given that overview indexing starts at 0, and the full resolution layer is not including in this numbering, this is the 4th layer in size of the file)

```
<Overview>
    <Dataset>another.tif</Dataset>
     <OpenOptions>
          <OOI key="OVERVIEW_LEVEL">2</OOI>
     </OpenOptions>
</Overview>
```

Example 5

This example points to another GTI dataset, with an explicit layer name, and by adjusting its extent to the one of the main GTI dataset (assuming such extent is 2,49,3,50).

```
<Overview>
    <Dataset>other.gti.gpkg</Dataset>
    <Layer>other_layer</Layer>
    <OpenOptions>
        <OOI key="MINX">2</OOI>
        <OOI key="MINY">49</OOI>
        <OOI key="MAXX">3</OOI>
        <OOI key="MAXY">50</OOI>
    </OpenOptions>
</Overview>
```

6.56.8.2 Overviews as metadata items of a GDAL vector layer

The recognized metadata items all start with `OVERVIEW_<idx>_` where idx is an integer index (starting at 0 since GDAL 3.9.2, starting at 1 in previous versions

- `OVERVIEW_<idx>_DATASET=<string>`

  Name of the dataset to use as an overview level. This may be a raster dataset (for example a GeoTIFF file, or another GTI dataset). This may also be a vector dataset with a GTI compatible layer, potentially specified with `OVERVIEW_<idx>_LAYER`.

  Starting with GDAL 3.9.2, overviews of `OVERVIEW_<idx>_DATASET=<string>` are also automatically added, unless `OVERVIEW_<idx>_OPEN_OPTIONS=OVERVIEW_LEVEL=NONE` is specified (or, since 3.13, `OVERVIEW_<idx>_FACTOR=1`)

- `OVERVIEW_<idx>_OPEN_OPTIONS=<key1=value1>[,key2=value2]...`

  Open options(s) to use to open `OVERVIEW_<idx>_DATASET`.

- `OVERVIEW_<idx>_LAYER=<string>`

  Only taken into account if `OVERVIEW_<idx>_DATASET=<string>` is not specified, or points to a GTI dataset.

  Name of the vector layer to use as an overview level, assuming `OVERVIEW_<idx>_DATASET` points to a vector dataset. `OVERVIEW_<idx>_DATASET` may also not be specified, in which case the vector dataset of the full resolution virtual mosaic is used.

- `OVERVIEW_<idx>_FACTOR=<int>`

  Sub-sampling factor, greater or equal to 1 (1 accepted only since GDAL 3.13)

  Before GDAL 3.13, was only taken into account if `OVERVIEW_<idx>_DATASET=<string>` was not not specified, or pointed to a GTI dataset. Since GDAL 3.13, `OVERVIEW_<idx>_FACTOR` is also accepted on any valid `OVERVIEW_<idx>_DATASET`.

  If `OVERVIEW_<idx>_DATASET` and `OVERVIEW_<idx>_LAYER` are not specified, then all tiles of the full resolution virtual mosaic are used, with the specified sub-sampling factor (it is recommended, but not required, that those tiles do have a corresponding overview). `OVERVIEW_<idx>_DATASET` and/or `OVERVIEW_<idx>_LAYER` may also be specified to point to another tile index.

Example 6

This example is the equivalent of all the examples of the *Overviews in XML* section.

```
# Use the sources of the current GTI file, downsampled by a factor of 2
OVERVIEW_0_FACTOR=2

# Use the full resolution of some.tif and all its potential overviews.
OVERVIEW_1_DATASET=some.tif

# Use the first overview level of another.tif that is at least smaller
# than the dimensions of its full resolution layer divided by a factor of 4.
OVERVIEW_2_DATASET=another.tif
OVERVIEW_2_FACTOR=4

# Use the overview of index 2 from another.tif
OVERVIEW_3_DATASET=another.tif
OVERVIEW_3_OPEN_OPTIONS=OVERVIEW_LEVEL=2

# Use 'other_layer' of 'other.gti.gpkg' and adjust its extent to the one
# of the main GTI dataset (assuming such extent is 2,49,3,50).
OVERVIEW_4_DATASET=other.gti.gpkg
OVERVIEW_4_LAYER=layer
OVERVIEW_4_OPEN_OPTIONS=MINX=2,MINY=49,MAXX=3,MAXY=50
```

6.56.9 Multi-threading optimizations

Starting with GDAL 3.10, the *NUM_THREADS* open option can be set to control specifically the multi-threading of GTI datasets. It defaults to `ALL_CPUS`, and when set, overrides *GDAL_NUM_THREADS* or *GTI_NUM_THREADS*. It applies to band-level and dataset-level RasterIO(), if more than 1 million pixels are requested and if the mosaic is made of only non-overlapping tiles.

- **NUM_THREADS=[integer/ALL_CPUS]:** Defaults to `ALL_CPUS`. Determines the number of threads used when an operation reads from multiple sources.

This can also be specified globally with the *GTI_NUM_THREADS* configuration option.

- **GTI_NUM_THREADS=[integer/ALL_CPUS]:** Defaults to `ALL_CPUS`. Determines the number of threads used when an operation reads from multiple sources.

Note that the number of threads actually used is also limited by the *GDAL_MAX_DATASET_POOL_SIZE* configuration option.

6.57 GTiff -- GeoTIFF File Format

Driver short name

GTiff

Driver built-in by default

This driver is built-in by default

Most forms of TIFF and GeoTIFF files are supported by GDAL for reading, and somewhat less varieties can be written.

GDAL also supports reading and writing BigTIFF files (evolution of the TIFF format to support files larger than 4 GB).

Currently band types of Byte, UInt16, Int16, UInt32, Int32, Float32, Float64, CInt16, CInt32, CFloat32 and CFloat64 are supported for reading and writing. Paletted images will return palette information associated with the band. The compression formats listed below should be supported for reading as well.

As well, one bit files, and some other unusual formulations of GeoTIFF file, such as YCbCr color model files, are automatically translated into RGBA (red, green, blue, alpha) form, and treated as four eight bit bands.

For an alternative which offers thread-safe read-only capabilities, consult *LIBERTIFF -- GeoTIFF File Format*.

6.57.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.57.2 Georeferencing

Most GeoTIFF projections should be supported, with the caveat that in order to translate uncommon Projected, and Geographic coordinate systems into OGC WKT it is necessary to have the PROJ proj.db database available. It must be found at the location pointed to by the PROJ_LIB environment variable, or at one of the locations set programmatically via OSRSetPROJSearchPaths().

Georeferencing from GeoTIFF is supported in the form of one tiepoint and pixel size, a transformation matrix, or a list of GCPs.

If no georeferencing information is available in the TIFF file itself, GDAL will also check for, and use an ESRI *world file* with the extension .tfw, .tifw/.tiffw or .wld, as well as a MapInfo .tab file.

By default, information is fetched in following order (first listed is the highest priority): PAM (Persistent Auxiliary metadata) .aux.xml sidecar file, INTERNAL (GeoTIFF keys and tags), TABFILE (.tab), WORLDFILE (.tfw, .tifw/.tiffw or .wld), XML (.xml)

The allowed sources and their priority order can be changed with the `GDAL_GEOREF_SOURCES` configuration option (or `GEOREF_SOURCES` open option) whose value is a comma-separated list of the following keywords : PAM, INTERNAL, TABFILE, WORLDFILE, XML (added in 3.7), NONE. Earlier mentioned sources take priority over later ones. A non mentioned source will be ignored.

For example setting it to "WORLDFILE,PAM,INTERNAL" will make a geotransformation matrix from a potential worldfile priority over PAM or GeoTIFF.

Minimum support for extracting the CRS from ESRI .xml side car files has been added in GDAL 3.7, using the metadata.refSysInfo.RefSystem.refSysID.identCode.code CRS code.

GDAL can read and write the *RPCCoefficientTag* as described in the RPCs in GeoTIFF proposed extension. The tag is written only for files created with the default profile GDALGeoTIFF. For other profiles, a .RPB file is created. In GDAL data model, the RPC coefficients are stored into the RPC metadata domain. For more details, see the rfc-22. If .RPB or _RPC.TXT files are found, they will be used to read the RPCs, even if the *RPCCoefficientTag* tag is set.

6.57.3 Internal nodata masks

TIFF files can contain internal transparency masks. The GeoTIFF driver recognizes an internal directory as being a transparency mask when the FILETYPE_MASK bit value is set on the TIFFTAG_SUBFILETYPE tag. According to the TIFF specification, such internal transparency masks contain 1 sample of 1-bit data. Although the TIFF specification allows for higher resolutions for the internal transparency mask, the GeoTIFF driver only supports internal transparency masks of the same dimensions as the main image. Transparency masks of internal overviews are also supported.

When the `GDAL_TIFF_INTERNAL_MASK` configuration option is set to YES (which is the case starting with GDAL 3.9 when CreateMaskBand() is invoked on the dataset, or when it is invoked on a raster band with the GMF_PER_DATASET flag), and the GeoTIFF file is opened in update mode, the CreateMaskBand() method on a TIFF dataset or rasterband will create an internal transparency mask. Otherwise, the default behavior of nodata mask creation will be used, that is to say the creation of a .msk file, as per rfc-15.

1-bit internal mask band are deflate compressed. When reading them back, to make conversion between mask band and alpha band easier, mask bands are exposed to the user as being promoted to full 8 bits (i.e. the value for unmasked pixels is 255) unless the `GDAL_TIFF_INTERNAL_MASK_TO_8BIT` configuration option is set to NO. This does not affect the way the mask band is written (it is always 1-bit).

6.57.4 Overviews

The GeoTIFF driver supports reading, creation and update of internal overviews. Internal overviews can be created on GeoTIFF files opened in update mode (with gdaladdo for instance). If the GeoTIFF file is opened as read only, the creation of overviews will be done in an external .ovr file. Overview are only updated on request with the BuildOverviews() method.

Several overview creation options are available. They can be provided through `GDALDataset::BuildOverviews()`, `GDALDataset::AddOverviews()` or with the `--creation-option` argument of *gdal raster overview add*.

Most of them can also be specified as configuration options by adding an `_OVERVIEW` suffix to the option name, but this is now considered a legacy behavior.

- **LOCATION=[INTERNAL/EXTERNAL/RRD]:** Set the location of the overview.
  - LOCATION=INTERNAL: only when the dataset to update is a (Geo)TIFF file.
  - LOCATION=EXTERNAL: implied when using option `--external` of *gdal raster overview add*
  - LOCATION=RRD: place the overviews in an associated .aux file suitable for direct use with Imagine or ArcGIS, as well as GDAL applications. Note that none of the below options apply when RRD is selected.
- **COMPRESS=[JPEG/LZW/PACKBITS/DEFLATE/CCITTRLE/CCITTFAX3/CCITTFAX4/LZMA/ZSTD/LERC/LERC_DEFLATE/LERC_ZSTD/WEBP/JXL/NONE]:** Set the compression to use. By default, `NONE`, unless the dataset to update is a (Geo)TIFF file, in which case the compression method of its full resolution layer will be used.
  - `JPEG` should generally only be used with Byte data (8 bit per channel). Better compression for RGB images can be obtained by using the PHOTOMETRIC=YCBCR colorspace with a 4:2:2 subsampling of the Y,Cb,Cr components.

  - CCITTFAX3, CCITTFAX4 or CCITRLE compression should only be used with 1bit (NBITS=1) data
  - LZW, DEFLATE and ZSTD compressions can be used with the PREDICTOR creation option.
  - ZSTD: available for GDAL builds (or libtiff when using external libtiff) against libzstd.
  - LERC and LERC_DEFLATE available for GDAL builds (or libtiff when using external liblerc) against liblerc.
  - LERC_ZSTD is available when LERC and ZSTD are available.
  - JXL is for JPEG-XL, and is only available when using internal libtiff and building GDAL against https://github.com/libjxl/libjxl . Supported data types are Byte, UInt16 and Float32 only.
- **BLOCKSIZE=<integer>:** Sets tile width and height. Must be divisible by 16 and a power-of-two value between 64 and 4096. By default, the block size of the full resolution layer (if tiled) is used, otherwise it defaults to 128. This can also be set with the GDAL_TIFF_OVR_BLOCKSIZE configuration option.
- **NUM_THREADS=[<integer>/ALL_CPUS]:** Defaults to 1. Enable multi-threaded compression by specifying the number of worker threads. Worthwhile for slow compression algorithms such as DEFLATE or LZMA. Will be ignored for JPEG. Default is compression in the main thread.
- **PREDICTOR=[1/2/3]:** Defaults to 1. Set the predictor for LZW, DEFLATE and ZSTD compression. The default is 1 (no predictor), 2 is horizontal differencing and 3 is floating point prediction. PREDICTOR=2 is only supported for 8, 16, 32 and 64 bit samples. PREDICTOR=3 is only supported for 16, 32 and 64 bit floating-point data.
- **JPEG_QUALITY=1-100:** Defaults to 75. Set the JPEG quality when using JPEG compression. Low values result in higher compression ratios, but poorer image quality with strong blocking artifacts. Values above 95 are not meaningfully better quality but can be substantially larger.
- **JPEGTABLESMODE=[0/1/2/3]:** Defaults to 1. Configure how and where JPEG quantization and Huffman tables are written in the TIFF JpegTables tag and strip/tile.
  - 0: JpegTables is not written. Each strip/tile contains its own quantization tables and use optimized Huffman coding.
  - 1: JpegTables is written with only the quantization tables. Each strip/tile refers to those quantized tables and use optimized Huffman coding. This is generally the optimal choice for smallest file size, and consequently is the default.
  - 2: JpegTables is written with only the default Huffman tables. Each strip/tile refers to those Huffman tables (thus no optimized Huffman coding) and contains its own quantization tables (identical). This option has no anticipated practical value.
  - 3: JpegTables is written with the quantization and default Huffman tables. Each strip/tile refers to those tables (thus no optimized Huffman coding). This option could perhaps with some data be more efficient than 1, but this should only occur in rare circumstances.
- **INTERLEAVE=[BAND/PIXEL]:** Control whether pixel or band interleaving is used.
- **PHOTOMETRIC=[MINISBLACK/MINISWHITE/RGB/CMYK/YCBCR/CIELAB/ICCLAB/ITULAB]:** Set the photometric interpretation tag. Default is MINISBLACK, but if the input image has 3 or 4 bands of Byte type, then RGB will be selected. You can override default photometric using this option.
- **ALPHA=[YES/NON-PREMULTIPLIED/PREMULTIPLIED/UNSPECIFIED]:** The first "extrasample" is marked as being alpha if there are any extra samples. This is necessary if you want to produce a greyscale TIFF file with an alpha band (for instance). YES is an alias for NON-PREMULTIPLIED alpha.
- **ZLEVEL=Integer between 1 and 9 (or 12 when libdeflate is used):** (GDAL >= 3.4.1) Defaults to 6. Deflate compression level, for COMPRESS=DEFLATE or LERC_DEFLATE.
- **ZSTD_LEVEL=Integer between 1 and 22:** (GDAL >= 3.4.1) Defaults to 9. ZSTD compression level, for COMPRESS=ZSTD or LERC_ZSTD.

- **WEBP_LEVEL=Integer between 0 and 100:** Defaults to `75`. WEBP quality level of overviews, either internal or external.
- **WEBP_LOSSLESS=[YES/NO]:** (GDAL >= 3.6) Defaults to `NO`. Whether WEBP compression is lossless or not.
- **MAX_Z_ERROR=value:** (GDAL >= 3.4.1) Defaults to `0` (lossless). Maximum error threshold on values for LERC/LERC_DEFLATE/LERC_ZSTD compression.
- **JXL_LOSSLESS=[YES/NO]:** Defaults to `YES`. Set whether JPEG-XL compression should be lossless (YES) or lossy (NO). For lossy compression, the underlying data should be either gray, gray+alpha, rgb or rgb+alpha. For lossy compression, the pixel data should span the whole range of the underlying pixel type (i.e. 0-255 for Byte, 0-65535 for UInt16)
- **JXL_EFFORT=[1-9]:** Defaults to `5`. Level of effort for JPEG-XL compression. The higher, the smaller file and slower compression time.
- **JXL_DISTANCE=[0.01-25]:** Defaults to `1.0`. (Only applies when JXL_LOSSLESS=NO) Distance level for lossy JPEG-XL compression. It is specified in multiples of a just-noticeable difference. (cf butteraugli for the definition of the distance) That is, 0 is mathematically lossless, 1 should be visually lossless, and higher distances yield denser and denser files with lower and lower fidelity. The recommended range is [0.5,3].
- **JXL_ALPHA_DISTANCE=[-1/0/[0.01-25]]:** Defaults to `-1`. (Only applies when JXL_LOSSLESS=NO) Requires libjxl > 0.8.1. Distance level for alpha channel for lossy JPEG-XL compression. It is specified in multiples of a just-noticeable difference. (cf butteraugli for the definition of the distance) That is, 0 is mathematically lossless, 1 should be visually lossless, and higher distances yield denser and denser files with lower and lower fidelity. For lossy compression, the recommended range is [0.5,3]. The default value is the special value -1.0, which means to use the same distance value as non-alpha channel (ie JXL_DISTANCE).
- **SPARSE_OK=[ON/OFF]:** Defaults to `OFF`. When set to ON, blocks whose pixels are all at nodata (or 0 if no nodata is defined) are not written in the file. Sparse files have 0 tile/strip offsets for blocks never written and save space; however, most non-GDAL packages cannot read such files.

6.57.5 Overviews and nodata masks

The following configurations can be encountered depending if overviews and nodata masks are internal or not.

- Internal overviews, internal nodata mask: If a GeoTIFF file has a internal transparency mask and the GeoTIFF file is opened in update mode, BuildOverviews() will automatically create overviews for the internal transparency mask.
- External overviews, external nodata mask: When opened in read-only mode, BuildOverviews() will automatically create overviews for the external transparency mask (in a .msk.ovr file)
- Internal overviews, external nodata mask: when running BuildOverviews() in update mode on the .tif file, starting with GDAL 3.9, overviews of the main bands and of the external .msk will be generated (as internal overview of the .msk). Prior to GDAL 3.9, only the overviews of the main bands are generated. Overviews of the external .msk file had to be explicitly generated by running BuildOverviews() on the .msk.

For the remaining configuration, behavior is less obvious:

- External overviews, internal nodata mask: when running BuildOverviews() in read-only mode on the .tif file, only the overviews of the main bands will be generated. Generating the overviews of the internal nodata mask is not currently supported by the driver.

Practical note: for a command line point of view, BuildOverviews() in update mode means "gdaladdo the.tiff" (without -ro). Whereas BuildOverviews() in read-only mode means "gdaladdo -ro the.tiff".

6.57.6 Metadata

GDAL can deal with the following baseline TIFF tags as dataset-level metadata :

- TIFFTAG_DOCUMENTNAME
- TIFFTAG_IMAGEDESCRIPTION
- TIFFTAG_SOFTWARE
- TIFFTAG_DATETIME
- TIFFTAG_ARTIST
- TIFFTAG_HOSTCOMPUTER
- TIFFTAG_COPYRIGHT
- TIFFTAG_XRESOLUTION
- TIFFTAG_YRESOLUTION
- TIFFTAG_RESOLUTIONUNIT
- TIFFTAG_MINSAMPLEVALUE (read only)
- TIFFTAG_MAXSAMPLEVALUE (read only)
- GEO_METADATA: This tag may be used for embedding XML-encoded instance documents prepared using 19139-based schema (GeoTIFF DGIWG)
- TIFF_RSID: This tag specifies a File Universal Unique Identifier, or RSID, according to DMF definition (GeoTIFF DGIWG)

The name of the metadata item to use is one of the above names ("TIFFTAG_DOCUMENTNAME", ...). On creation, those tags can for example be set with

```
gdal_translate in.tif out.tif -mo {TAGNAME}=VALUE
```

Other non standard metadata items can be stored in a TIFF file created with the profile GDALGeoTIFF (the default, see below in the Creation issues section). Those metadata items are grouped together into a XML string stored in the non standard TIFFTAG_GDAL_METADATA ASCII tag (code 42112). When BASELINE or GeoTIFF profile are used, those non standard metadata items are stored into a PAM .aux.xml file.

The value of GDALMD_AREA_OR_POINT ("AREA_OR_POINT") metadata item is stored in the GeoTIFF key RasterPixelIsPoint for GDALGeoTIFF or GeoTIFF profiles.

XMP metadata can be extracted from the file, and will be reported as XML raw content in the xml:XMP metadata domain.

EXIF metadata can be extracted from the file, and will be reported in the EXIF metadata domain.

6.57.7 Color Profile Metadata

GDAL can deal with the following color profile metadata in the COLOR_PROFILE domain:

- SOURCE_ICC_PROFILE (Base64 encoded ICC profile embedded in file. If available, other tags are ignored.)
- SOURCE_PRIMARIES_RED (xyY in "x,y,1" format for red primary.)
- SOURCE_PRIMARIES_GREEN (xyY in "x,y,1" format for green primary)
- SOURCE_PRIMARIES_BLUE (xyY in "x,y,1" format for blue primary)
- SOURCE_WHITEPOINT (xyY in "x,y,1" format for whitepoint)
- TIFFTAG_TRANSFERFUNCTION_RED (Red table of TIFFTAG_TRANSFERFUNCTION)

- TIFFTAG_TRANSFERFUNCTION_GREEN (Green table of TIFFTAG_TRANSFERFUNCTION)
- TIFFTAG_TRANSFERFUNCTION_BLUE (Blue table of TIFFTAG_TRANSFERFUNCTION)
- TIFFTAG_TRANSFERRANGE_BLACK (Min range of TIFFTAG_TRANSFERRANGE)
- TIFFTAG_TRANSFERRANGE_WHITE (Max range of TIFFTAG_TRANSFERRANGE)

Note that these metadata properties can only be used on the original raw pixel data. If automatic conversion to RGB has been done, the color profile information cannot be used.

All these metadata tags can be overridden and/or used as creation options.

6.57.8 json:ISIS3 Metadata

Added in version 3.13.

It is possible to use `GDALDataset::GetMetadataItem()` to request one particular key in the `json:ISIS3` metadata domain among all the top-level keys that would be returned by `GDALDataset::GetMetadata("json:ISIS3")`.

6.57.9 Nodata value

GDAL stores band nodata value in the non standard TIFFTAG_GDAL_NODATA ASCII tag (code 42113) for files created with the default profile GDALGeoTIFF. Note that all bands must use the same nodata value. When BASELINE or GeoTIFF profile are used, the nodata value is stored into a PAM .aux.xml file.

6.57.10 Raster Attribute Table

Starting with GDAL 3.11, raster attribute tables stored in auxiliary .tif.vat.dbf files, as written by ArcGIS, can be read as GDAL Raster Attribute Table.

Starting with GDAL 3.12, raster attribute tables can be written into and read from the `GDAL_METADATA` TIFF tag. If the *GTIFF_WRITE_RAT_TO_PAM* configuration option is set, raster attribute tables will be written instead into the `.aux.xml` side car file.

6.57.11 Sparse files

GDAL makes a special interpretation of a TIFF tile or strip whose offset and byte count are set to 0, that is to say a tile or strip that has no corresponding allocated physical storage. On reading, such tiles or strips are considered to be implicitly set to 0 or to the nodata value when it is defined. On writing, it is possible to enable generating such files through the Create() interface by setting the *SPARSE_OK* creation option to YES. Then, blocks that are never written through the IWriteBlock()/IRasterIO() interfaces will have their offset and byte count set to 0. This is particularly useful to save disk space and time when the file must be initialized empty before being passed to a further processing stage that will fill it. To avoid ambiguities with another sparse mechanism discussed in the next paragraphs, we will call such files with implicit tiles/strips "TIFF sparse files". They will be likely **not** interoperable with TIFF readers that are not GDAL based and would consider such files with implicit tiles/strips as defective.

This mechanism is extended to the CreateCopy() and Open() interfaces (for update mode) as well. If the *SPARSE_OK* creation option (or the *SPARSE_OK* open option for Open()) is set to YES, even an attempt to write a all 0/nodata block will be detected so that the tile/strip is not allocated (if it was already allocated, then its content will be replaced by the 0/nodata content).

In the case where *SPARSE_OK* is **not** defined (or set to its default value FALSE), for uncompressed files whose nodata value is not set, or set to 0, in Create() and CreateCopy() mode, the driver will delay the allocation of 0-blocks until file closing, so as to be able to write them at the very end of the file, and in a way compatible with the filesystem sparse file mechanisms (to be distinguished from the TIFF sparse file extension discussed earlier). That is that all the empty blocks will be seen as properly allocated from the TIFF point of view (corresponding strips/tiles will have valid offsets and byte counts), but will have no corresponding physical storage. Provided that the filesystem supports such sparse

files, which is the case for most Linux popular filesystems (ext2/3/4, xfs, btfs, ...) or NTFS on Windows. If the file system does not support sparse files, physical storage will be allocated and filled with zeros.

6.57.12 Raw mode

For some TIFF formulations that have "odd" photometric color spaces, on-the-fly decoding as RGBA is done. This might not be desirable in some use cases. This behavior can be disabled by prefixing the filename with GTIFF_RAW:

For example to translate a CMYK file to another one :

```
gdal_translate GTIFF_RAW:in.tif out.tif -co PHOTOMETRIC=CMYK
```

6.57.13 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). This driver supports the following open options:

NUM_THREADS=[<number_of_threads>/ALL_CPUS]: Enable multi-threaded compression by specifying the number of worker threads. Worth it for slow compression algorithms such as DEFLATE or LZMA. Default is compression in the main thread. Starting with GDAL 3.6, this option also enables multi-threaded decoding when RasterIO() requests intersect several tiles/strips. The *GDAL_NUM_THREADS* configuration option can also be used as an alternative to setting the open option.

GEOREF_SOURCES=comma-separated list with one or several of PAM, INTERNAL, TABFILE, WORLDFILE or XML (XML added in 3.7): Define which georeferencing sources are allowed and their priority order. See *Georeferencing* paragraph.

SPARSE_OK=[TRUE/FALSE]: Defaults to `FALSE`. Should empty blocks be omitted on disk? When this option is set, any attempt of writing a block whose all pixels are 0 or the nodata value will cause it not to be written at all (unless there is a corresponding block already allocated in the file). Sparse files have 0 tile/strip offsets for blocks never written and save space; however, most non-GDAL packages cannot read such files.

IGNORE_COG_LAYOUT_BREAK=[YES/NO]: (GDAL >= 3.8) Defaults to `NO`. Updating a COG (Cloud Optimized GeoTIFF) file generally breaks part of the optimizations, but still produces a valid GeoTIFF file. Starting with GDAL 3.8, to avoid undesired loss of the COG characteristics, opening such a file in update mode will be rejected, unless this option is also set to YES (default is NO). This option has only effect on COG files and when opening in update mode, and is ignored on regular (Geo)TIFF files.

COLOR_TABLE_MULTIPLIER=[AUTO/1/256/257]: (GDAL >= 3.10.0) Specifies the value by which to multiply GDAL color table entry values, usually in [0,255] range, to obtain a TIFF color map value, or on reading by which to divide TIFF color map values to get a GDAL color table entry. GDAL consistently uses 257, but it might be necessary to use 256 for compatibility with files generated by other software. In AUTO mode, GDAL 3.10 or later can automatically detect the 256 multiplication factor when all values in the TIFF color map are multiple of that value.

6.57.14 Creation Issues

GeoTIFF files can be created with any GDAL defined band type, including the complex types. Created files may have any number of bands. Files of type Byte with exactly 3 bands will be given a photometric interpretation of RGB, files of type Byte with exactly four bands will have a photometric interpretation of RGBA, while all other combinations will have a photometric interpretation of MIN_IS_BLACK. Non-standard (regarding to the intrinsics TIFF capabilities) band color interpretation, such as BGR ordering, will be handled in creation and reading, by storing them in the GDAL internal metadata TIFF tag.

The TIFF format only supports R,G,B components for palettes / color tables. Thus on writing the alpha information will be silently discarded.

6.57.14.1 Creation Options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). This driver supports the following creation options:

- **TFW=[YES/NO]:** Force the generation of an associated ESRI world file (.tfw). See the *World Files* page for details.
- **RPB=[YES/NO]:** Force the generation of an associated .RPB file to describe RPC (Rational Polynomial Coefficients), if RPC information is available. If not specified, this file is automatically generated if there's RPC information and that the PROFILE is not the default GDALGeoTIFF.
- **RPCTXT=[YES/NO]:** Force the generation of an associated _RPC.TXT file to describe RPC (Rational Polynomial Coefficients), if RPC information is available.
- **INTERLEAVE=[BAND/PIXEL]:** Set the interleaving to use. See *Multiband pixel organization (INTERLEAVE metadata item)* for more details.
  - `PIXEL`: for each spatial block, one TIFF tile/strip gathering values for all bands is used . This matches the `contiguous` planar configuration in TIFF terminology. This is also known as a `BIP (Band Interleaved Per Pixel)` organization. Such method is the default, and may be slightly less efficient than BAND interleaving for some purposes, but some applications only support pixel interleaved TIFF files. On the other hand, image-based compression methods may perform better using PIXEL interleaving. For JPEG PHOTOMETRIC=YCbCr, pixel interleaving is required. It is also required for WebP compression.

    Assuming a pixel[band][y][x] indexed array, when using CreateCopy(), the pseudo code for the file disposition is:

    ```
    for y in 0 ... numberOfBlocksInHeight - 1:
        for x in 0 ... numberOfTilesInWidth - 1:
            for j in 0 ... blockHeight - 1:
                for i in 0 ... blockWidth -1:
                    start_new_strip_or_tile()
                    for band in 0 ... numberBands -1:
                        write(pixel[band][y*blockHeight+j][x*blockWidth+i])
                    end_new_strip_or_tile()
                end_for
            end_for
        end_for
    end_for
    ```

  - `BAND`: for each spatial block, one TIFF tile/strip is used for each band. This matches the contiguous `separate` configuration in TIFF terminology. This is also known as a `BSQ (Band SeQuential)` organization.

    In addition to that, when using CreateCopy(), data for the first band is written first, followed by data for the second band, etc. The pseudo code for the file disposition is:

    ```
    for y in 0 ... numberOfBlocksInHeight - 1:
        for x in 0 ... numberOfTilesInWidth - 1:
            start_new_strip_or_tile()
            for band in 0 ... numberBands -1:
                for j in 0 ... blockHeight - 1:
                    for i in 0 ... blockWidth -1:
                        write(pixel[band][y*blockHeight+j][x*blockWidth+i])
                    end_for
                end_for
    ```

(continues on next page)

(continued from previous page)

```
        end_new_strip_or_tile()
    end_for
end_for
```

Starting with GDAL 3.5, when copying from a source dataset with multiple bands which advertises a INTERLEAVE metadata item, if the INTERLEAVE creation option is not specified, the source dataset INTERLEAVE will be automatically taken into account, unless the COMPRESS creation option is specified.

> **Note**
>
> Starting with GDAL 3.11, a `TILE` interleaving is available, but only when using the *COG -- Cloud Optimized GeoTIFF generator* driver.

- **TILED=[YES/NO]:** Defaults to `NO`. By default striped TIFF files are created. This option can be used to force creation of tiled TIFF files.
- **BLOCKXSIZE=<integer>:** Defaults to `256`. Sets tile width. Must be divisible by 16.
- **BLOCKYSIZE=<integer>:** Set tile or strip height. Tile height defaults to 256, strip height defaults to a value such that one strip is 8K or less. Must be divisible by 16 when *TILED*=YES.
- **NBITS=<value>:** Create a file with less than 8 bits per sample by passing a value from 1 to 7. The apparent pixel type should be Byte. Values of n=9...15 (UInt16 type) and n=17...31 (UInt32 type) are also accepted. n=16 is accepted for Float32 type to generate half-precision floating point values.
- **COMPRESS=[JPEG/LZW/PACKBITS/DEFLATE/CCITTRLE/CCITTFAX3/CCITTFAX4/LZMA/ZSTD/LERC/LERC_DEFLATE/LERC_ZSTD/WEBP/JXL/NONE]:** Set the compression to use.
  - `JPEG` should generally only be used with Byte data (8 bit per channel). Better compression for RGB images can be obtained by using the PHOTOMETRIC=YCBCR colorspace with a 4:2:2 subsampling of the Y,Cb,Cr components.

    Starting with GDAL 3.4, if GDAL is built with its internal libtiff, read and write support for JPEG-in-TIFF compressed images with 12-bit sample is enabled by default (if JPEG support is also enabled), using GDAL internal libjpeg (based on IJG libjpeg-6b, with additional changes for 12-bit sample support). Support for JPEG with 12-bit sample is independent of whether 8-bit JPEG support is enabled through internal IJG libjpeg-6b or external libjpeg (like libjpeg-turbo)
  - `CCITTFAX3`, `CCITTFAX4` or `CCITRLE` compression should only be used with 1bit (NBITS=1) data
  - `LZW`, `DEFLATE` and `ZSTD` compressions can be used with the PREDICTOR creation option.
  - `ZSTD` is available when using internal libtiff and if GDAL built against libzstd >=1.0, or if built against external libtiff with zstd support.
  - `LERC` and `LERC_DEFLATE` are available only when using internal libtiff for GDAL < 3.3.0. Since GDAL 3.3.0, LERC compression is also available when building GDAL against external libtiff >= 4.3.0, built itself against https://github.com/esri/lerc
  - `LERC_ZSTD` is available when `LERC` and `ZSTD` are available.
  - `JXL` is for JPEG-XL, and is only available when using internal libtiff and building GDAL against https://github.com/libjxl/libjxl . It is recommended to use JXL compression with the `TILED=YES` creation option and block size of 256x256, 512x512, or 1024x1024 pixels. Supported data types are `Byte`, `UInt16` and `Float32` only. For GDAL < 3.6.0, JXL compression may only be used alongside `INTERLEAVE=PIXEL` (the default) on datasets with 4 bands or less.
  - `NONE` is the default.

- **NUM_THREADS=[<integer>/ALL_CPUS]:** Defaults to 1. Enable multi-threaded compression by specifying the number of worker threads. Worthwhile for slow compression algorithms such as DEFLATE or LZMA. Will be ignored for JPEG. Default is compression in the main thread.
- **PREDICTOR=[1/2/3]:** Defaults to 1. Set the predictor for LZW, DEFLATE and ZSTD compression. The default is 1 (no predictor), 2 is horizontal differencing and 3 is floating point prediction. PREDICTOR=2 is only supported for 8, 16, 32 and 64 bit samples (support for 64 bit was added in libtiff > 4.3.0). PREDICTOR=3 is only supported for 16, 32 and 64 bit floating-point data.
- **DISCARD_LSB=[<nbits>/nbits_band1,nbits_band2,...nbits_bandN]:** Set the number of least-significant bits to clear, possibly different per band. Lossy compression scheme to be best used with PREDICTOR=2 and LZW/DEFLATE/ZSTD compression.
- **SPARSE_OK=[TRUE/FALSE]:** Should newly created files (through Create() interface) be allowed to be sparse? Sparse files have 0 tile/strip offsets for blocks never written and save space; however, most non-GDAL packages cannot read such files. SPARSE_OK=TRUE is also supported through the CreateCopy() interface. Even an attempt to write a block whose all pixels are 0 or the nodata value will cause it not to be written at all (unless there is a corresponding block already allocated in the file). The default is FALSE.
- **JPEG_QUALITY=1-100:** Defaults to 75. Set the JPEG quality when using JPEG compression. Low values result in higher compression ratios, but poorer image quality with strong blocking artifacts. Values above 95 are not meaningfully better quality but can be substantially larger.
- **JPEGTABLESMODE=[0/1/2/3]:** Defaults to 1. Configure how and where JPEG quantization and Huffman tables are written in the TIFF JpegTables tag and strip/tile.
  - 0: JpegTables is not written. Each strip/tile contains its own quantization tables and use optimized Huffman coding.
  - 1: JpegTables is written with only the quantization tables. Each strip/tile refers to those quantized tables and use optimized Huffman coding. This is generally the optimal choice for smallest file size, and consequently is the default.
  - 2: JpegTables is written with only the default Huffman tables. Each strip/tile refers to those Huffman tables (thus no optimized Huffman coding) and contains its own quantization tables (identical). This option has no anticipated practical value.
  - 3: JpegTables is written with the quantization and default Huffman tables. Each strip/tile refers to those tables (thus no optimized Huffman coding). This option could perhaps with some data be more efficient than 1, but this should only occur in rare circumstances.
- **ZLEVEL=[1-9] or [1-12]:** Defaults to 6. Set the level of compression when using DEFLATE compression (or LERC_DEFLATE). A value of 9 (resp. 12) is best/slowest when using zlib (resp. libdeflate), and 1 is least/fastest compression.
- **ZSTD_LEVEL=[1-22]:** Defaults to 9. Set the level of compression when using ZSTD compression (or LERC_ZSTD). A value of 22 is best (very slow), and 1 is least compression.
- **MAX_Z_ERROR=<threshold>:** Defaults to 0. Set the maximum error threshold on values for LERC/LERC_DEFLATE/LERC_ZSTD compression. The default is 0 (lossless).
- **MAX_Z_ERROR_OVERVIEW=<threshold>:** (GDAL >= 3.8) Set the maximum error threshold on values for LERC/LERC_DEFLATE/LERC_ZSTD compression, on overviews. The default is the value of *MAX_Z_ERROR*
- **WEBP_LEVEL=[1-100]:** Defaults to 75. Set the WEBP quality level when using WEBP compression. A value of 100 is best quality (least compression), and 1 is worst quality (best compression).
- **WEBP_LOSSLESS=[TRUE/FALSE]:** Requires libwebp >= 0.1.4. By default, lossy compression is used. If set to True, lossless compression will be used. There is a significant time penalty for each tile/strip with lossless WebP compression, so you may want to increase the *BLOCKYSIZE* value for strip layout.

- **JXL_LOSSLESS=[YES/NO]:** Defaults to `YES`. Set whether JPEG-XL compression should be lossless (YES) or lossy (NO). For lossy compression, the underlying data should be either gray, gray+alpha, rgb or rgb+alpha. For lossy compression, the pixel data should span the whole range of the underlying pixel type (i.e. 0-255 for Byte, 0-65535 for UInt16)
- **JXL_EFFORT=[1-9]:** Defaults to `5`. Level of effort for JPEG-XL compression. The higher, the smaller file and slower compression time.
- **JXL_DISTANCE=[0.01-25]:** Defaults to `1.0`. (Only applies when JXL_LOSSLESS=NO) Distance level for lossy JPEG-XL compression. It is specified in multiples of a just-noticeable difference. (cf butteraugli for the definition of the distance) That is, 0 is mathematically lossless, 1 should be visually lossless, and higher distances yield denser and denser files with lower and lower fidelity. The recommended range is [0.5,3].
- **JXL_ALPHA_DISTANCE=[-1/0/[0.01-25]]:** (GDAL >= 3.7) Defaults to `-1`. (Only applies when JXL_LOSSLESS=NO) Requires libjxl > 0.8.1. Distance level for alpha channel for lossy JPEG-XL compression. It is specified in multiples of a just-noticeable difference. (cf butteraugli for the definition of the distance) That is, 0 is mathematically lossless, 1 should be visually lossless, and higher distances yield denser and denser files with lower and lower fidelity. For lossy compression, the recommended range is [0.5,3]. The default value is the special value -1.0, which means to use the same distance value as non-alpha channel (ie JXL_DISTANCE).
- **PHOTOMETRIC=[MINISBLACK/MINISWHITE/RGB/CMYK/YCBCR/CIELAB/ICCLAB/ITULAB]:** Set the photometric interpretation tag. Default is MINISBLACK, but if the input image has 3 or 4 bands of Byte type, then RGB will be selected. You can override default photometric using this option.
- **ALPHA=[YES/NON-PREMULTIPLIED/PREMULTIPLIED/UNSPECIFIED]:** The first "extrasample" is marked as being alpha if there are any extra samples. This is necessary if you want to produce a greyscale TIFF file with an alpha band (for instance). YES is an alias for NON-PREMULTIPLIED alpha.
- **PROFILE=[GDALGeoTIFF/GeoTIFF/BASELINE]:** Defaults to `GDALGeoTIFF`. Control what non-baseline tags are emitted by GDAL.
  - With `GDALGeoTIFF` (the default) various GDAL custom tags may be written.
  - With `GeoTIFF` only GeoTIFF tags will be added to the baseline.
  - With `BASELINE` no GDAL or GeoTIFF tags will be written. BASELINE is occasionally useful when writing files to be read by applications intolerant of unrecognized tags.
- **BIGTIFF=[YES/NO/IF_NEEDED/IF_SAFER]:** Defaults to `IF_NEEDED`. Control whether the created file is a BigTIFF or a classic TIFF.
  - `YES` forces BigTIFF.
  - `NO` forces classic TIFF.
  - `IF_NEEDED` will only create a BigTIFF if it is clearly needed (in the uncompressed case, and image larger than 4GB. So no effect when using a compression).
  - `IF_SAFER` will create BigTIFF if the resulting file *might* exceed 4GB. Note: this is only a heuristic that might not always work depending on compression ratios.

  BigTIFF is a TIFF variant which can contain more than 4GiB of data (size of classic TIFF is limited by that value). This option is available if GDAL is built with libtiff library version 4.0 or higher.

  When creating a new GeoTIFF with no compression, GDAL computes in advance the size of the resulting file. If that computed file size is over 4GiB, GDAL will automatically decide to create a BigTIFF file. However, when compression is used, it is not possible in advance to known the final size of the file, so classical TIFF will be chosen. In that case, the user must explicitly require the creation of a BigTIFF with *`BIGTIFF=YES`* if the final file is anticipated to be too big for classical TIFF format. If BigTIFF creation is not explicitly asked or guessed and the resulting file is too big for classical TIFF, libtiff will fail with an error message like "TIFFAppendToStrip:Maximum TIFF file size exceeded".

- **PIXELTYPE=[DEFAULT/SIGNEDBYTE]:** By setting this to `SIGNEDBYTE`, a new Byte file can be forced to be written as signed byte. Starting with GDAL 3.7, this option is deprecated and Int8 should rather be used.
- **COPY_SRC_OVERVIEWS=[YES/NO]:** Defaults to `NO`. Used by (*`GDALDriver::CreateCopy()`* only) By setting this to YES, the potential existing overviews of the source dataset will be copied to the target dataset without being recomputed. This option is typically used to generate Cloud Optimized Geotiff (starting with GDAL 3.1, the *COG -- Cloud Optimized GeoTIFF generator* driver can be used as a convenient shortcut). If overviews of mask band also exist, provided that the *`GDAL_TIFF_INTERNAL_MASK`* configuration option is set to YES (which is the case starting with GDAL 3.9), they will also be copied. Note that this creation option will have no effect if general options (i.e. options which are not creation options) of gdal_translate are used. Creation options related to compression are also applied to the output overviews.
- **STREAMABLE_OUTPUT=[YES/NO]:** Generate a file with an Image File Directory and block order that allow streaming. See *Streaming operations*.
- **GEOTIFF_KEYS_FLAVOR=[STANDARD/ESRI_PE]:** Defaults to `STANDARD`. Determine which "flavor" of GeoTIFF keys must be used to write the SRS information. The STANDARD way (default choice) will use the general accepted formulations of GeoTIFF keys, including extensions of the values accepted for ProjectedCSTypeGeoKey to new EPSG codes. The ESRI_PE flavor will write formulations that are (more) compatible with ArcGIS. At the time of writing, the ESRI_PE choice has mostly an effect when writing the EPSG:3857 (Web Mercator) SRS. For other SRS, the standard way will be used, with the addition of a ESRI_PE WKT string as the value of PCSCitationGeoKey.
- **GEOTIFF_VERSION=[AUTO/1.0/1.1]:** (GDAL >= 3.1.0) Defaults to `AUTO`. Select the version of the GeoTIFF standard used to encode georeferencing information. `1.0` corresponds to the original 1995, GeoTIFF Revision 1.0, by Ritter & Ruth. `1.1` corresponds to the OGC standard 19-008, which is an evolution of 1.0, which clear ambiguities and fix inconsistencies mostly in the processing of the vertical part of a CRS. `AUTO` mode (default value) will generally select 1.0, unless the CRS to encode has a vertical component or is a 3D CRS, in which case 1.1 is used.

  > **Note**
  >
  > Write support for GeoTIFF 1.1 requires libgeotiff 1.6.0 or later.

- **COLOR_TABLE_MULTIPLIER=[1/256/257]:** (GDAL >= 3.10.0) Defaults to `257`. Specifies the value by which to multiply GDAL color table entry values, usually in [0,255] range, to obtain a TIFF color map value. GDAL consistently uses 257, but it might be necessary to use 256 for compatibility with files generated by other software.

6.57.14.2 Subdatasets

Multi-page TIFF files are exposed as subdatasets. On opening, a subdataset name is GTIFF_DIR:{index}:filename.tif, where {index} starts at 1.

Starting with GDAL 3.0, subdataset creation is possible by using the APPEND_SUBDATASET=YES creation option. The filename passed to Create() / CreateCopy() should be the regular filename (not with GTIFF_DIR: syntax. Creating overviews on a multi-page TIFF is not supported.

Starting with GDAL 3.2, read-only access to subdataset overviews and masks is possible provided that they are referenced by their parent IFD through the TIFFTAG_SUBIFD tag.

6.57.14.3 About JPEG compression of RGB images

When translating a RGB image to JPEG-In-TIFF, using PHOTOMETRIC=YCBCR can make the size of the image typically 2 to 3 times smaller than the default photometric value (RGB). When using PHOTOMETRIC=YCBCR, the INTERLEAVE option must be kept to its default value (PIXEL), otherwise libtiff will fail to compress the data.

Note also that the dimensions of the tiles or strips must be a multiple of 8 for PHOTOMETRIC=RGB or 16 for PHOTOMETRIC=YCBCR

6.57.14.4 Lossless conversion of JPEG into JPEG-in-TIFF

The conversion of a JPEG file (but *not* a JPEG-in-TIFF file) to a JPEG-in-TIFF file without decompression and compression cycles, and thus without any additional quality loss, can be done with gdal_translate (or the CreateCopy() API), if all the following conditions are met:

- the source dataset is a JPEG file (or a VRT with a JPEG as a single SimpleSource)
- the target dataset is a JPEG-in-TIFF file
- no explicitly target JPEG quality is specified
- no change in colorspace is specified
- no sub-windowing is requested
- and more generally, no change that alters pixel values

The generation of a tiled JPEG-in-TIFF from the original JPEG image is possible. Explicit assignment of target SRS and bounds are also possible.

So, the following commands will use the lossless copy method :

```
gdal_translate in.jpg out.tif -co COMPRESS=JPEG

gdal_translate in.jpg out.tif -co COMPRESS=JPEG -co TILED=YES

gdal_translate in.jpg out.tif -co COMPRESS=JPEG -a_srs EPSG:4326 -a_ullr -180 90 180 -90
```

whereas the following commands will *not* (and thus cause JPEG decompression and compression):

```
gdal_translate in.jpg out.tif -co COMPRESS=JPEG -co JPEG_QUALITY=60

gdal_translate in.jpg out.tif -srcwin 0 0 500 500 -co COMPRESS=JPEG
```

6.57.14.5 Streaming operations

The GeoTIFF driver can support reading or writing TIFF files (with some restrictions detailed below) in a streaming compatible way.

When reading a file from /vsistdin/, a named pipe (on Unix), or if forcing streamed reading by setting the *TIFF_READ_STREAMING* configuration option to YES, the GeoTIFF driver will assume that the TIFF Image File Directory (IFD) is at the beginning of the file, i.e. at offset 8 for a classical TIFF file or at offset 16 for a BigTIFF file. The values of the tags of array type must be contained at the beginning of file, after the end of the IFD and before the first image strip/tile. The reader must read the strips/tiles in the order they are written in the file. For a pixel interleaved file (PlanarConfiguration=Contig), the recommended order for a writer, and thus for a reader, is from top to bottom for a strip-organized file or from top to bottom, which a chunk of a block height, and left to right for a tile-organized file. For a band organized file (PlanarConfiguration=Separate), the above order is recommended with the content of the first band, then the content of the second band, etc... Technically this order corresponds to increasing offsets in the TileOffsets/StripOffsets tag. This is the order that the GDAL raster copy routine will assume.

If the order is not the one described above, the UNORDERED_BLOCKS=YES dataset metadata item will be set in the TIFF metadata domain. Each block offset can be determined by querying the "BLOCK_OFFSET_[xblock]_[yblock]" band metadata items in the TIFF metadata domain (where xblock, yblock is the coordinate of the block), and a reader could use that information to determine the appropriate reading order for image blocks.

The files that are streamed into the GeoTIFF driver may be compressed, even if the GeoTIFF driver cannot produce such files in streamable output mode (regular creation of TIFF files will produce such compatible files for streamed reading).

When writing a file to /vsistdout/, a named pipe (on Unix), or when defining the *STREAMABLE_OUTPUT=YES* creation option, the CreateCopy() method of the GeoTIFF driver will generate a file with the above defined constraints (related to position of IFD and block order), and this is only supported for a uncompressed file. The Create() method also supports creating streamable compatible files, but the writer must be careful to set the projection, geotransform or metadata before writing image blocks (so that the IFD is written at the beginning of the file). And when writing image blocks, the order of blocks must be the one of the above paragraph, otherwise errors will be reported.

Some examples :

```
gdal_translate in.tif /vsistdout/ -co TILED=YES | gzip | gunzip | gdal_translate /
↪vsistdin/ out.tif -co TILED=YES -co COMPRESS=DEFLATE
```

or

```
mkfifo my_fifo
gdalwarp in.tif my_fifo -t_srs EPSG:3857
gdal_translate my_fifo out.png -of PNG
```

Note: not all utilities are compatible with such input or output streaming operations, and even those which may deal with such files may not manage to deal with them in all circumstances, for example if the reading driver driven by the output file is not compatible with the block order of the streamed input.

6.57.14.6 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). This paragraph lists the configuration options that can be set to alter the default behavior of the GTiff driver.

- **GTIFF_IGNORE_READ_ERRORS=[TRUE/FALSE]:** Can be set to TRUE to avoid turning libtiff errors into GDAL errors. Can help reading partially corrupted TIFF files.
- **ESRI_XML_PAM=[TRUE/FALSE]:** Can be set to TRUE to force metadata in the xml:ESRI domain to be written to PAM.
- **BIGTIFF_OVERVIEW=value:** Control whether BigTIFF should be used for overviews. Options are the same as *BIGTIFF* creation option.
- **COMPRESS_OVERVIEW=value:** See *COMPRESS* creation option. Set the compression type to use for overviews. For internal overviews, only honoured since GDAL 3.6.
- **INTERLEAVE_OVERVIEW=[BAND/PIXEL]:** Control whether pixel or band interleaving is used for overviews.
- **PHOTOMETRIC_OVERVIEW=value:** Set the photometric color space for overview creation
- **PREDICTOR_OVERVIEW=[1/2/3]:** Set the predictor to use for overviews with LZW, DEFLATE and ZSTD compression.
- **JPEG_QUALITY_OVERVIEW=Integer between 0 and 100:** Defaults to `75`. Quality of JPEG compressed overviews, either internal or external.

- **WEBP_LEVEL_OVERVIEW=Integer between 0 and 100:** Defaults to `75`. WEBP quality level of overviews, either internal or external.
- **WEBP_LOSSLESS_OVERVIEW=[YES/NO]:** (GDAL >= 3.6) Defaults to `NO`. Whether WEBP compression is lossless or not.
- **ZLEVEL_OVERVIEW=Integer between 1 and 9 (or 12 when libdeflate is used):** (GDAL >= 3.4.1) Defaults to `6`. Deflate compression level of overviews, for COMPRESS_OVERVIEW=DEFLATE or LERC_DEFLATE, either internal or external.
- **ZSTD_LEVEL_OVERVIEW=Integer between 1 and 22:** (GDAL >= 3.4.1) Defaults to `9`. ZSTD compression level of overviews, for COMPRESS_OVERVIEW=ZSTD or LERC_ZSTD, either internal or external.
- **MAX_Z_ERROR_OVERVIEW=value:** (GDAL >= 3.4.1) Defaults to `0` (lossless). Maximum error threshold on values for LERC/LERC_DEFLATE/LERC_ZSTD compression of overviews, either internal or external.
- **SPARSE_OK_OVERVIEW=[ON/OFF]:** (GDAL >= 3.4.1) Defaults to `OFF`. When set to ON, blocks whose pixels are all at nodata (or 0 if no nodata is defined) are not written in the file. Sparse files have 0 tile/strip offsets for blocks never written and save space; however, most non-GDAL packages cannot read such files.
- **GDAL_TIFF_INTERNAL_MASK=[TRUE/FALSE]:** Defaults to `TRUE` (since GDAL 3.9), FALSE (GDAL 3.8 or earlier). See *Internal nodata masks* section
- **GDAL_TIFF_INTERNAL_MASK_TO_8BIT=[TRUE/FALSE]:** Defaults to `TRUE`. See *Internal nodata masks* section.
- *USE_RRD*: Can be set to TRUE to force external overviews in the RRD format.
- **TIFF_USE_OVR=[TRUE/FALSE]:** Defaults to `FALSE`. Can be set to TRUE to force external overviews in the GeoTIFF (.ovr) format.
- **TIFF_READ_STREAMING=[YES/NO]:** If `YES`, assume that Image File Directory is at the beginning of the file. See *Streaming operations*.
- **GTIFF_POINT_GEO_IGNORE=[TRUE/FALSE]:** Defaults to `FALSE`. Can be set to TRUE to revert back to the behavior of ancient GDAL versions regarding how PixelIsPoint is interpreted w.r.t geotransform. See rfc-33 for more details.
- **GTIFF_REPORT_COMPD_CS=[TRUE/FALSE]:** Can be set to TRUE to avoid stripping the vertical CRS of compound CRS when reading the SRS of a file. Does not affect the writing side. Default value : FALSE for GeoTIFF 1.0 files, or TRUE (starting with GDAL 3.1) for GeoTIFF 1.1 files.
- **GDAL_ENABLE_TIFF_SPLIT=[TRUE/FALSE]:** Defaults to `TRUE`. Can be set to FALSE to avoid all-in-one-strip files being presented as having.
- **GDAL_TIFF_OVR_BLOCKSIZE=value:** See *Overviews* section.
- **GTIFF_LINEAR_UNITS=value:** Can be set to BROKEN to read GeoTIFF files that have false easting/northing improperly set in meters when it ought to be in coordinate system linear units. (Ticket #3901).
- **TAB_APPROX_GEOTRANSFORM=[YES/NO]:** Defaults to `NO`. To decide if an approximate geotransform is acceptable when reading a .tab file.
- **GTIFF_DIRECT_IO=[YES/NO]:** Defaults to `NO`. Can be set to YES to use specialized RasterIO() implementations when reading un-compressed TIFF files to avoid using the block cache. Setting it to YES even when the optimized cases do not apply should be safe (generic implementation will be used).
- **GTIFF_VIRTUAL_MEM_IO=[YES/NO/IF_ENOUGH_RAM]:** Defaults to `NO`. Can be set to YES to use specialized RasterIO() implementations when reading un-compressed TIFF files to avoid using the block cache. This implementation relies on memory-mapped file I/O, and is currently only supported on Linux (64-bit build strongly recommended) or, on other POSIX-like systems. Setting it to YES even when the optimized cases do not apply should be safe (generic implementation will be used), but if the file exceeds RAM, disk swapping might occur if the whole file is read. Setting it to IF_ENOUGH_RAM will first check if the uncompressed file size is

no bigger than the physical memory. If both *GTIFF_VIRTUAL_MEM_IO* and *GTIFF_DIRECT_IO* are enabled, the former is used in priority, and if not possible, the later is tried.

- *GDAL_NUM_THREADS* enables multi-threaded compression by specifying the number of worker threads. Worth it for slow compression algorithms such as DEFLATE or LZMA. Will be ignored for JPEG. Default is compression in the main thread. Note: this configuration option also apply to other parts to GDAL (warping, gridding, ...). Starting with GDAL 3.6, this option also enables multi-threaded decoding when RasterIO() requests intersect several tiles/strips.
- **GTIFF_WRITE_RAT_TO_PAM=[YES/NO]:** (GDAL >= 3.12.0) Defaults to `YES`. Whether raster attribute tables attached to bands of a GeoTIFF dataset must be serialized in the `.aux.xml` side-car file, instead of the `GDAL_METADATA` TIFF tag. In versions prior to 3.12, raster attribute tables were always written and read in the `.aux.xml` side-car file.
- **GTIFF_WRITE_TOWGS84=[AUTO/YES/NO]:** (GDAL >= 3.0.3) When set to AUTO, a GeogTOWGS84GeoKey geokey will be written with TOWGS84 3 or 7-parameter Helmert transformation, if the CRS has no EPSG code attached to it, or if the TOWGS84 transformation attached to the CRS doesn't match the one imported from the EPSG code. If set to YES, then the TOWGS84 transformation attached to the CRS will be always written. If set to NO, then the transformation will not be written in any situation.
- **CHECK_DISK_FREE_SPACE=[YES/NO]:** Defaults to `YES`. If `YES`, ensure that enough disk space is available before attempting to write an uncompressed, non-sparse GeoTIFF. Disabling this check is not expected to be necessary, unless GDAL is incorrectly determining the disk space available on the destination file system.
- **GTIFF_READ_ANGULAR_PARAMS_IN_DEGREE=[YES/NO]:** (GDAL >= 3.9.1) Defaults to `NO`. Conformant GeoTIFF files should have the values of angular projection parameters written in the unit of the GeogAngularUnitsGeoKey. But some non-conformant implementations, such as GDAL <= 3.9.0, always wrote them in degrees. This option can be set to YES when reading such non-conformant GeoTIFF files (typically using grads), to instruct GDAL (>= 3.9.1) that the projection parameters are in degrees, instead of being expressed in the unit of the GeogAngularUnitsGeoKey.
- **GTIFF_WRITE_ANGULAR_PARAMS_IN_DEGREE=[YES/NO]:** (GDAL >= 3.9.1) Defaults to `NO`. Conformant GeoTIFF files should have the values of angular projection parameters written in the unit of the GeogAngularUnitsGeoKey. But some non-conformant implementations, such as GDAL >= 3.0 and <= 3.9.0, assumed those values to be in degree. This option can be set to YES to force writing such non-conformant GeoTIFF files. It should *not* be nominally used, except to workaround interoperability issues.

6.57.15 Codec Recommendations

6.57.15.1 LZW

If you don't know what to choose, choose this one.

6.57.15.2 DEFLATE

The most commonly supported TIFF codec, especially with older non-geo software.

6.57.15.3 LERC

Used for storing quantized floating point data. https://github.com/esri/lerc

6.57.15.4 ZSTD

Smaller and faster than DEFLATE, but not as commonly supported.

6.57.15.5 WEBP

A smaller and faster JPEG.

6.57.15.6 JXL

Next-gen JPG from the JPG group.

6.57.15.7 LZMA

Slow but storage efficient.

6.57.15.8 CCITTRLE/CCITTFAX3/CCITTFAX4

Did you happen to have fax files from the 1990s? Use these.

6.57.16 See Also

- GeoTIFF Information Page
- libtiff Page
- Details on BigTIFF file format
- *COG -- Cloud Optimized GeoTIFF generator* driver

6.57.16.1 WLD -- ESRI World File

A world file file is a plain ASCII text file consisting of six values separated by newlines. The format is:

```
pixel X size
rotation about the Y axis (usually 0.0)
rotation about the X axis (usually 0.0)
negative pixel Y size
X coordinate of upper left pixel center
Y coordinate of upper left pixel center
```

For example:

```
60.0000000000
0.0000000000
0.0000000000
-60.0000000000
440750.0000000000
3751290.0000000000
```

You can construct that file simply by using your favorite text editor.

World file usually has suffix `.wld`, but sometimes it may has `.tfw`, `.tifw`, `.jgw` or other suffixes depending on the image file it comes with.

6.58 GXF -- Grid eXchange File

Driver short name

GXF

Driver built-in by default

This driver is built-in by default

This is a raster exchange format propagated by Geosoft, and made a standard in the gravity/magnetics field. GDAL supports reading (but not writing) GXF-3 files, including support for georeferencing information, and projections.

By default, the datatype returned for GXF datasets by GDAL is Float32. You can specify the datatype by setting the GXF_DATATYPE configuration option (Float64 supported currently)

Details on the supporting code, and format can be found on the GXF-3 page.

NOTE: Implemented as gxfdataset.cpp.

6.58.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.59 GTX — NOAA Vertical Datum Grid Shift

Driver short name

GTX

Driver built-in by default

This driver is built-in by default

The GTX format is used to store vertical datum grid shift grids, typically distributed by NOAA. These grids have been used historically by PROJ, until PROJ 7.0 where GeoTIFF-based grids have been introduced and are now the preferred way to convey geoid models.

6.59.1 Driver capabilities

- Read-only
- Raster

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.59.2 File format

GTX files are binary raster grids containing vertical offset values. They are commonly used in conjunction with coordinate reference systems that include vertical components.

6.59.3 See also

- *gdalwarp*
- https://proj.org/

6.60 HDF4 -- Hierarchical Data Format Release 4 (HDF4)

Driver short name

HDF4

Driver short name

HDF4Image

Build dependencies

libdf

There are two HDF formats, HDF4 (4.x and previous releases) and HDF5. These formats are completely different and NOT compatible. This driver intended for HDF4 file formats importing. NASA's Earth Observing System (EOS) maintains its own HDF modification called HDF-EOS. This modification is suited for use with remote sensing data and fully compatible with underlying HDF. This driver can import HDF4-EOS files. Currently EOS use HDF4-EOS for data storing (telemetry from "Terra" and "Aqua" satellites). In the future they will switch to HDF5-EOS format, which will be used for telemetry from "Aura" satellite.

6.60.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

6.60.2 Multiple Image Handling (Subdatasets)

Hierarchical Data Format is a container for several different datasets. For data storing Scientific Datasets (SDS) used most often. SDS is a multidimensional array filled by data. One HDF file may contain several different SDS arrays. They may differ in size, number of dimensions and may represent data for different regions.

If the file contains only one SDS that appears to be an image, it may be accessed normally, but if it contains multiple images it may be necessary to import the file via a two step process. The first step is to get a report of the components images (SDS arrays) in the file using **gdalinfo**, and then to import the desired images using gdal_translate. The **gdalinfo** utility lists all multidimensional subdatasets from the input HDF file. The name of individual images (subdatasets) are assigned to the **SUBDATASET_n_NAME** metadata item. The description for each image is found in the **SUBDATASET_n_DESC** metadata item. For HDF4 images the subdataset names will be formatted like this:

HDF4_SDS:subdataset_type:file_name:subdataset_index

where *subdataset_type* shows predefined names for some of the well known HDF datasets, *file_name* is the name of the input file, and *subdataset_index* is the index of the image to use (for internal use in GDAL).

On the second step you should provide this name for **gdalinfo** or **gdal_translate** for actual reading of the data.

For example, we want to read data from the MODIS Level 1B dataset:

```
$ gdalinfo GSUB1.A2001124.0855.003.200219309451.hdf
Driver: HDF4/Hierarchical Data Format Release 4
Size is 512, 512
Coordinate System is `'
Metadata:
  HDFEOSVersion=HDFEOS_V2.7
  Number of Scans=204
  Number of Day mode scans=204
  Number of Night mode scans=0
  Incomplete Scans=0
```

...a lot of metadata output skipped...

```
Subdatasets:
  SUBDATASET_1_NAME=HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:0
  SUBDATASET_1_DESC=[408x271] Latitude (32-bit floating-point)
```

(continues on next page)

(continued from previous page)

```
  SUBDATASET_2_NAME=HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:1
  SUBDATASET_2_DESC=[408x271] Longitude (32-bit floating-point)
  SUBDATASET_3_NAME=HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:2
  SUBDATASET_3_DESC=[12x2040x1354] EV_1KM_RefSB (16-bit unsigned integer)
  SUBDATASET_4_NAME=HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:3
  SUBDATASET_4_DESC=[12x2040x1354] EV_1KM_RefSB_Uncert_Indexes (8-bit unsigned integer)
  SUBDATASET_5_NAME=HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:4
  SUBDATASET_5_DESC=[408x271] Height (16-bit integer)
  SUBDATASET_6_NAME=HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:5
  SUBDATASET_6_DESC=[408x271] SensorZenith (16-bit integer)
  SUBDATASET_7_NAME=HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:6
  SUBDATASET_7_DESC=[408x271] SensorAzimuth (16-bit integer)
  SUBDATASET_8_NAME=HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:7
  SUBDATASET_8_DESC=[408x271] Range (16-bit unsigned integer)
  SUBDATASET_9_NAME=HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:8
  SUBDATASET_9_DESC=[408x271] SolarZenith (16-bit integer)
  SUBDATASET_10_NAME=HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:9
  SUBDATASET_10_DESC=[408x271] SolarAzimuth (16-bit integer)
  SUBDATASET_11_NAME=HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:10
  SUBDATASET_11_DESC=[408x271] gflags (8-bit unsigned integer)
  SUBDATASET_12_NAME=HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:12
  SUBDATASET_12_DESC=[16x10] Noise in Thermal Detectors (8-bit unsigned integer)
  SUBDATASET_13_NAME=HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:13
  SUBDATASET_13_DESC=[16x10] Change in relative responses of thermal detectors (8-bit␣
↪unsigned integer)
  SUBDATASET_14_NAME=HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:14
  SUBDATASET_14_DESC=[204x16x10] DC Restore Change for Thermal Bands (8-bit integer)
  SUBDATASET_15_NAME=HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:15
  SUBDATASET_15_DESC=[204x2x40] DC Restore Change for Reflective 250m Bands (8-bit␣
↪integer)
  SUBDATASET_16_NAME=HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:16
  SUBDATASET_16_DESC=[204x5x20] DC Restore Change for Reflective 500m Bands (8-bit␣
↪integer)
  SUBDATASET_17_NAME=HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:17
  SUBDATASET_17_DESC=[204x15x10] DC Restore Change for Reflective 1km Bands (8-bit␣
↪integer)
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0,  512.0)
Upper Right (  512.0,    0.0)
Lower Right (  512.0,  512.0)
Center      (  256.0,  256.0)
```

Now select one of the subdatasets, described as `[12x2040x1354] EV_1KM_RefSB (16-bit unsigned integer)`:

```
$ gdalinfo HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:2
Driver: HDF4Image/HDF4 Internal Dataset
Size is 1354, 2040
Coordinate System is `'
Metadata:
  long_name=Earth View 1KM Reflective Solar Bands Scaled Integers
```

...metadata skipped...

```
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0, 2040.0)
Upper Right ( 1354.0,    0.0)
Lower Right ( 1354.0, 2040.0)
Center      (  677.0, 1020.0)
Band 1 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
Band 2 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
Band 3 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
Band 4 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
Band 5 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
Band 6 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
Band 7 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
Band 8 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
Band 9 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
Band 10 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
Band 11 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
Band 12 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
```

Or you may use **gdal_translate** for reading image bands from this dataset.

Note that you should provide exactly the contents of the line marked **SUBDATASET_n_NAME** to GDAL, including the **HDF4_SDS:** prefix.

This driver is intended only for importing remote sensing and geospatial datasets in form of raster images. If you want explore all data contained in HDF file you should use another tools (you can find information about different HDF tools using links at end of this page).

6.60.3 Georeference

There is no universal way of storing georeferencing in HDF files. However, some product types have mechanisms for saving georeferencing, and some of these are supported by GDAL. Currently supported are (*subdataset_type* shown in parenthesis):

- HDF4 files created by GDAL (**GDAL_HDF4**)
- ASTER Level 1A (**ASTER_L1A**)
- ASTER Level 1B (**ASTER_L1B**)
- ASTER Level 2 (**ASTER_L2**)
- ASTER DEM (**AST14DEM**)
- MODIS Level 1B Earth View products (**MODIS_L1B**)
- MODIS Level 3 products (**MODIS_L3**)
- SeaWiFS Level 3 Standard Mapped Image Products (**SEAWIFS_L3**)

By default the hdf4 driver only reads the gcps from every 10th row and column from EOS_SWATH datasets. You can change this behavior by setting the GEOL_AS_GCPS environment variable to PARTIAL (default), NONE, or FULL.

6.60.4 Creation Issues

This driver supports creation of the HDF4 Scientific Datasets. You may create set of 2D datasets (one per each input band) or single 3D dataset where the third dimension represents band numbers. All metadata and band descriptions from the input dataset are stored as HDF4 attributes. Projection information (if it exists) and affine transformation coefficients also stored in form of attributes. Files, created by GDAL have the special attribute:

"Signature=Created with GDAL (http://www.remotesensing.org/gdal/)"

and are automatically recognised when read, so the projection info and transformation matrix restored back.

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following creation options are supported:

RANK=[2/3]: Defaults to 3. Create **n**-dimensional SDS. Currently only 2D and 3D datasets supported. By default a 3-dimensional dataset will be created.

6.60.5 Metadata

All HDF4 attributes are transparently translated as GDAL metadata. In the HDF file attributes may be assigned assigned to the whole file as well as to particular subdatasets.

6.60.6 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open option is supported:

LIST_SDS=[AUTO/YES/NO]: (GDAL >= 3.2) Whether to report Scientific Data Sets (SDS). By default, when a HDF file contains EOS_SWATH or EOS_GRID, SDS will not be listed as GDAL subdatasets (as this would cause them to be reported twice). Listing them can be forced by setting LIST_SDS to YES.

6.60.7 Multidimensional API support

Added in version 3.1.

The HDF4 driver supports the *Multidimensional Raster Data Model* for reading operations.

6.60.8 Driver building

This driver built on top of NCSA HDF library, so you need one to compile GDAL with HDF4 support. You may search your operating system distribution for the precompiled binaries or download source code or binaries from the NCSA HDF Home Page (see links below).

Please note, that NCSA HDF library compiled with several defaults which is defined in *hlimits.h* file. For example, *hlimits.h* defines the maximum number of opened files:

```
#   define MAX_FILE   32
```

If you need open more HDF4 files simultaneously you should change this value and rebuild HDF4 library (and relink GDAL if using static HDF libraries).

6.60.9 See Also

- Implemented as hdf4dataset.cpp and hdf4imagedataset.cpp.
- The HDF Group
- Sources of the data in HDF4 and HDF4-EOS formats: Earth Observing System Data Gateway

Documentation to individual products, supported by this driver:

- Geo-Referencing ASTER L1B Data
- ASTER Standard Data Product Specifications Document
- MODIS Level 1B Product Information and Status
- MODIS Ocean User's Guide

6.61 HDF5 -- Hierarchical Data Format Release 5 (HDF5)

Driver short name

HDF5

Driver short name

HDF5Image

Build dependencies

libhdf5

This driver intended for HDF5 file formats importing.

Starting with GDAL 3.7, the driver can parse HDF-EOS5 encoded grid and swath metadata to establish projection and geotransform (for grids), and geolocation array (for swaths)

6.61.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.61.2 Multiple Image Handling (Subdatasets)

Hierarchical Data Format is a container for several different datasets. For data storing. HDF contains multidimensional arrays filled by data. One HDF file may contain several arrays. They may differ in size, number of dimensions.

The first step is to get a report of the components images (arrays) in the file using **gdalinfo**, and then to import the desired images using gdal_translate. The **gdalinfo** utility lists all multidimensional subdatasets from the input HDF file. The name of individual images (subdatasets) are assigned to the **SUBDATASET_n_NAME** metadata item. The description for each image is found in the **SUBDATASET_n_DESC** metadata item. For HDF5 images the subdataset names will be formatted like this:

HDF5:file_name:subdataset

where:
file_name is the name of the input file, and
subdataset is the dataset name of the array to use (for internal use in GDAL).

Starting with GDAL 3.10, specifying the `-if HDF5` option to command line utilities accepting it, or `HDF5` as the only value of the `papszAllowedDrivers` of *GDALOpenEx()*, also forces the driver to recognize the passed filename, when it is not using subdataset syntax (it can typically be used to force open a netCDF file that would be nominally recognized by the netCDF driver).

On the second step you should provide this name for **gdalinfo** or **gdal_translate** for actual reading of the data.

For example, we want to read data from the OMI/Aura Ozone (O3) dataset:

```
$ gdalinfo OMI-Aura_L2-OMTO3_2005m0326t2307-o03709_v002-2005m0428t201311.he5
Driver: HDF5/Hierarchical Data Format Release 5
Size is 512, 512
Coordinate System is `'

Subdatasets:
  SUBDATASET_1_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/APrioriLayerO3
  SUBDATASET_1_DESC=[1496x60x11] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/
↪APrioriLayerO3 (32-bit floating-point)
  SUBDATASET_2_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/AlgorithmFlags
  SUBDATASET_2_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/
↪AlgorithmFlags (8-bit unsigned character)
  SUBDATASET_3_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/CloudFraction
  SUBDATASET_3_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/
↪CloudFraction (32-bit floating-point)
  SUBDATASET_4_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/CloudTopPressure
  SUBDATASET_4_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/
↪CloudTopPressure (32-bit floating-point)
  SUBDATASET_5_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/ColumnAmountO3
  SUBDATASET_5_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/
↪ColumnAmountO3 (32-bit floating-point)
  SUBDATASET_6_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/LayerEfficiency
  SUBDATASET_6_DESC=[1496x60x11] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/
↪LayerEfficiency (32-bit floating-point)
  SUBDATASET_7_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/NValue
  SUBDATASET_7_DESC=[1496x60x12] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/NValue␣
↪(32-bit floating-point)
  SUBDATASET_8_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/O3BelowCloud
  SUBDATASET_8_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/
↪O3BelowCloud (32-bit floating-point)
  SUBDATASET_9_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/QualityFlags
  SUBDATASET_9_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/
↪QualityFlags (16-bit unsigned integer)
  SUBDATASET_10_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/Reflectivity331
  SUBDATASET_10_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/
```

(continues on next page)

(continued from previous page)

```
↪Reflectivity331 (32-bit floating-point)
  SUBDATASET_11_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/Reflectivity360
  SUBDATASET_11_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/
↪Reflectivity360 (32-bit floating-point)
  SUBDATASET_12_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/Residual
  SUBDATASET_12_DESC=[1496x60x12] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/
↪Residual (32-bit floating-point)
  SUBDATASET_13_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/ResidualStep1
  SUBDATASET_13_DESC=[1496x60x12] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/
↪ResidualStep1 (32-bit floating-point)
  SUBDATASET_14_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/ResidualStep2
  SUBDATASET_14_DESC=[1496x60x12] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/
↪ResidualStep2 (32-bit floating-point)
  SUBDATASET_15_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/SO2index
  SUBDATASET_15_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/SO2index␣
↪(32-bit floating-point)
  SUBDATASET_16_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/Sensitivity
  SUBDATASET_16_DESC=[1496x60x12] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/
↪Sensitivity (32-bit floating-point)
  SUBDATASET_17_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/StepOneO3
  SUBDATASET_17_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/
↪StepOneO3 (32-bit floating-point)
  SUBDATASET_18_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/StepTwoO3
  SUBDATASET_18_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/
↪StepTwoO3 (32-bit floating-point)
  SUBDATASET_19_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/TerrainPressure
  SUBDATASET_19_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/
↪TerrainPressure (32-bit floating-point)
  SUBDATASET_20_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/UVAerosolIndex
  SUBDATASET_20_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/
↪UVAerosolIndex (32-bit floating-point)
  SUBDATASET_21_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/dN_dR
  SUBDATASET_21_DESC=[1496x60x12] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/dN_dR␣
↪(32-bit floating-point)
  SUBDATASET_22_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/dN_dT
  SUBDATASET_22_DESC=[1496x60x12] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/dN_dT␣
↪(32-bit floating-point)
  SUBDATASET_23_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/GroundPixelQualityFlags
  SUBDATASET_23_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/
```

(continues on next page)

(continued from previous page)

```
↪GroundPixelQualityFlags (16-bit unsigned integer)
  SUBDATASET_24_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/Latitude
  SUBDATASET_24_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/
↪Latitude (32-bit floating-point)
  SUBDATASET_25_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/Longitude
  SUBDATASET_25_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/
↪Longitude (32-bit floating-point)
  SUBDATASET_26_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/RelativeAzimuthAngle
  SUBDATASET_26_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/
↪RelativeAzimuthAngle (32-bit floating-point)
  SUBDATASET_27_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/SolarAzimuthAngle
  SUBDATASET_27_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/
↪SolarAzimuthAngle (32-bit floating-point)
  SUBDATASET_28_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/SolarZenithAngle
  SUBDATASET_28_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/
↪SolarZenithAngle (32-bit floating-point)
  SUBDATASET_29_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/TerrainHeight
  SUBDATASET_29_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/
↪TerrainHeight (16-bit integer)
  SUBDATASET_30_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/ViewingAzimuthAngle
  SUBDATASET_30_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/
↪ViewingAzimuthAngle (32-bit floating-point)
  SUBDATASET_31_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.
↪he5":/HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/ViewingZenithAngle
  SUBDATASET_31_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/
↪ViewingZenithAngle (32-bit floating-point)
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0,  512.0)
Upper Right (  512.0,    0.0)
Lower Right (  512.0,  512.0)
Center      (  256.0,  256.0)
```

Now select one of the subdatasets, described as `[1645x60] CloudFraction (32-bit floating-point)`:

```
$ gdalinfo HDF5:"OMI-Aura_L2-OMTO3_2005m0326t2307-o03709_v002-2005m0428t201311.he5
↪":CloudFraction
Driver: HDF5Image/HDF5 Dataset
Size is 60, 1645
Coordinate System is:
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        TOWGS84[0,0,0,0,0,0,0],
```

(continues on next page)

(continued from previous page)

```
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0,
        AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.0174532925199433,
        AUTHORITY["EPSG","9108"]],
    AXIS["Lat",NORTH],
    AXIS["Long",EAST],
    AUTHORITY["EPSG","4326"]]
GCP Projection = GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.
↪257223563,AUTHORITY["EPSG","7030"]],TOWGS84[0,0,0,0,0,0,0],AUTHORITY["EPSG","6326"]],
↪PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT["degree",0.0174532925199433,
↪AUTHORITY["EPSG","9108"]],AXIS["Lat",NORTH],AXIS["Long",EAST],AUTHORITY["EPSG","4326"]]
GCP[  0]: Id=, Info=
          (0.5,0.5) -> (261.575,-84.3495,0)
GCP[  1]: Id=, Info=
          (2.5,0.5) -> (240.826,-85.9928,0)
GCP[  2]: Id=, Info=
          (4.5,0.5) -> (216.754,-86.5932,0)
GCP[  3]: Id=, Info=
          (6.5,0.5) -> (195.5,-86.5541,0)
GCP[  4]: Id=, Info=
          (8.5,0.5) -> (180.265,-86.2009,0)
GCP[  5]: Id=, Info=
          (10.5,0.5) -> (170.011,-85.7315,0)
GCP[  6]: Id=, Info=
          (12.5,0.5) -> (162.987,-85.2337,0)
... 3000 GCPs are read from the file if Latitude and Longitude arrays are presents
```

Corner Coordinates: Upper Left (0.0, 0.0) Lower Left (0.0, 1645.0) Upper Right (60.0, 0.0) Lower Right (60.0, 1645.0) Center (30.0, 822.5) Band 1 Block=60x1 Type=Float32, ColorInterp=Undefined Open GDAL Datasets: 1 N DriverIsNULL 512x512x0

You may use **gdal_translate** for reading image bands from this dataset.

Note that you should provide exactly the contents of the line marked **SUBDATASET_n_NAME** to GDAL, including the **HDF5:** prefix.

This driver is intended only for importing remote sensing and geospatial datasets in form of raster images(2D or 3D arrays). If you want explore all data contained in HDF file you should use another tools (you can find information about different HDF tools using links at end of this page).

6.61.3 Georeference

There is no universal way of storing georeferencing in HDF files. However, some product types have mechanisms for saving georeferencing, and some of these are supported by GDAL. Currently supported are (*subdataset_type* shown in parenthesis):

- HDF5 OMI/Aura Ozone (O3) Total Column 1-Orbit L2 Swath 13x24km (**Level-2 OMTO3**)
- HDF-EOS5 grids (starting with GDAL 3.7)
- HDF-EOS5 swaths (starting with GDAL 3.7)

6.61.4 Multi-file support

Starting with GDAL 3.1, the driver supports opening datasets split over several files using the 'family' HDF5 file driver. For that, GDAL must be provided with the filename of the first part, containing in it a single '0' (zero) character, or ending with 0.h5 or 0.hdf5

6.61.5 Multidimensional API support

Added in version 3.1.

The HDF5 driver supports the *Multidimensional Raster Data Model* for reading operations.

6.61.6 Driver building

This driver built on top of NCSA HDF5 library, so you need to download prebuild HDF5 libraries: HDF5-1.6.4 library or higher. You also need zlib 1.2 and szlib 2.0. For windows user be sure to set the attributes writable (especially if you are using Cygwin) and that the DLLs can be located somewhere by your PATH environment variable. You may also download source code NCSA HDF Home Page (see links below).

6.61.7 See Also

Implemented as hdf5dataset.cpp and hdf5imagedataset.cpp.

The NCSA HDF5 Download Page at the National Center for Supercomputing Applications

The HDFView is a visual tool for browsing and editing NCSA HDF4 and HDF5 files.

Documentation to individual products, supported by this driver:

- OMTO3: OMI/Aura Ozone (O3) Total Column 1-Orbit L2 Swath 13x24km V003
- HDF-EOS5 Data Model, File Format and Library

6.62 HEIF / HEIC -- ISO/IEC 23008-12 High Efficiency Image File Format

Added in version 3.2.

Driver short name

HEIF

Build dependencies

libheif (>= 1.1), built against libde265

High Efficiency Image File Format (HEIF) is a container format for individual images and image sequences. The driver has mostly been developed and tested to be able to read images using the High Efficiency Video Coding (HEVC, ITU-T H.265) codec. Such images are usually called HEIC (HEVC in HEIF) files, and have the .heic extension. iOS 11 can generate such files.

libheif 1.4 or later is needed to support images with more than 8-bits per channel.

Later versions of libheif may also support one or more of AVIF (AV1 in HEIF), JPEG, JPEG 2000 and uncompressed images depending on compile-time options.

The driver can read EXIF metadata (exposed in the `EXIF` metadata domain) and XMP metadata (exposed in the `xml:XMP` metadata domain)

The driver will expose the thumbnail as an overview (when its number of bands matches the number of bands in the full resolution image).

If a file contains several top-level images, they will be exposed as GDAL subdatasets.

6.62.1 AVIF support

Starting with GDAL 3.10, the AVIF_HEIF companion driver to the HEIF driver may be used to open images encoding with the AVIF (AV1 Image File) codec if the *AVIF -- AV1 Image File Format* driver is not available and if libheif has been compiled with support for one of the libraries it support that are able of AV1 decoding (libaom or libdav1d).

6.62.2 Driver capabilities

Supports VirtualIO

if libheif >= 1.4

Supports CreateCopy()

This driver supports the *`GDALDriver::CreateCopy()`* operation

Supports Georeferencing

This driver supports georeferencing

6.62.3 Georeferencing

HEIF provides experimental read-only support for GeoHEIF internal referencing, based on the draft OGC 24-038 design. There is no support for writing at this time.

This requires at least libheif 1.19.0.

6.62.4 Built hints on Windows

- Download source archives for libheif at https://github.com/strukturag/libheif and libde265 at https://github.com/strukturag/libde265
- Unpack the archives (for example libde265-1.0.5.tar.gz and libheif-1.7.0.tar.gz)
- Build libde265:

```
cd libde265-1.0.5
mkdir build
cd build
cmake -G "Visual Studio 15 2017 Win64" .. -DCMAKE_INSTALL_PREFIX=c:/dev/
↪install-libheif
cmake --build . --config Release --target install
cd ..
copy libde265\de265.h c:/dev/install-libheif/include/libde265
```

(continues on next page)

(continued from previous page)

```
copy libde265\de265-version.h c:/dev/install-libheif/include/libde265
cd ..
```

- Build libheif with libde265 support:

```
cd libheif-1.7.0
mkdir build
cd build
cmake -G "Visual Studio 15 2017 Win64" .. \
    -DCMAKE_INSTALL_PREFIX=c:/dev/install-libheif \
    -DLIBDE265_FOUND=ON \
    -DLIBDE265_CFLAGS=/Ic:/dev/install-libheif/include \
    -DLIBDE265_LIBRARIES=c:/dev/install-libheif/lib/libde265
```

- Add in GDAL's nmake.local the following lines before building GDAL:

```
HEIF_INC = -Ic:\dev\install-libheif\include
HEIF_LIB = C:\dev\install-libheif\lib\heif.lib
```

6.63 HF2 -- HF2/HFZ heightfield raster

Driver short name

HF2

Driver built-in by default

This driver is built-in by default

GDAL supports reading and writing HF2/HFZ/HF2.GZ heightfield raster datasets.

HF2 is a heightfield format that records difference between consecutive cell values. HF2 files can also optionally be compressed by the gzip algorithm, and so the HF2.GZ files (or HFZ, equivalently) may be significantly smaller than the uncompressed data. The file format enables the user to have control on the desired accuracy through the vertical precision parameter.

GDAL can read and write georeferencing information through extended header blocks.

6.63.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.63.2 Creation options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following creation options are supported:

- **COMPRESS=[YES/NO]:** Defaults to `NO`. Whether the file must be compressed with GZip.
- **BLOCKSIZE=value:** Defaults to `256`. Internal tile size. Must be >= 8.
- **VERTICAL_PRECISION=value:** Defaults to `0.01`. Vertical precision. Must be > 0. Increasing the vertical precision decreases the file size, especially with *`COMPRESS=YES`*, but at the loss of accuracy.

6.63.3 See also

- Specification of HF2/HFZ format
- Specification of HF2 extended header blocks

6.64 HFA -- Erdas Imagine .img

Driver short name

HFA

Driver built-in by default

This driver is built-in by default

GDAL supports Erdas Imagine .img format for read access and write. The driver supports reading overviews, palettes, and georeferencing. It supports the Erdas band types u8, s8, u16, s16, u32, s32, f32, f64, c64 and c128.

Compressed and missing tiles in Erdas files should be handled properly on read. Files between 2GiB and 4GiB in size should work on Windows NT, and may work on some Unix platforms. Files with external spill files (needed for datasets larger than 2GiB) are also supported for reading and writing.

Metadata reading and writing is supported at the dataset and band level. But this is GDAL specific metadata, not metadata in an Imagine recognized form. The metadata is stored in a table called GDAL_MetaData which as many columns as metadata items. The title of the column is the `key` of the `key``=``value` metadata item pair and the value of row 1 is `value`.

6.64.1 Driver capabilities

Supports CreateCopy()

This driver supports the *`GDALDriver::CreateCopy()`* operation

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.64.2 Creation Issues

Erdas Imagine files can be created with any GDAL defined band type, including the complex types. Created files may have any number of bands. Pseudo-Color tables will be written if using the GDALDriver::CreateCopy() methodology. Most projections should be supported, though translation of unusual datums (other than WGS84, WGS72, NAD83, and NAD27) may be problematic.

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following creation options are supported:

- **BLOCKSIZE=32-2048:** Defaults to `64`. Tile width/height.
- **USE_SPILL=[YES/NO]:** Defaults to `NO`. Force the generation of a spill file (by default spill file created for images larger 2GiB only).
- **COMPRESSED=[YES/NO]:** Defaults to `NO`. Create file as compressed. Use of spill file disables compression.
- **NBITS=[1/2/4]:** Create file with special sub-byte data types.
- **PIXELTYPE=[DEFAULT/SIGNEDBYTE]:** By setting this to SIGNEDBYTE, a new Byte file can be forced to be written as signed byte. Starting with GDAL 3.7, this option is deprecated and Int8 should rather be used.
- **AUX=[YES/NO]:** Defaults to `NO`. To create a .aux file.
- **IGNOREUTM=[YES/NO]:** Defaults to `NO`. Ignore UTM when selecting coordinate system - will use Transverse Mercator. Only used for Create() method.
- **STATISTICS=[YES/NO]:** Defaults to `NO`. To generate statistics and a histogram.
- **DEPENDENT_FILE=<filename>:** Name of dependent file (must not have absolute path). Optional
- **FORCETOPESTRING=[YES/NO]:** Defaults to `NO`. Force the use of ESRI Projection Engine (PE) String in file instead of Imagine coordinate system format. In some cases this improves ArcGIS coordinate system compatibility.
- **DISABLEPESTRING=[YES/NO]:** (GDAL >= 3.7) Defaults to `NO`. Disable use of ESRI Projection Engine (PE) String in file. The default value is NO, allowing the ESRI PE String to be written if needed.

Erdas Imagine supports external creation of overviews (with gdaladdo for instance). To force them to be created in an .rrd file (rather than inside the original .img) set the global config option HFA_USE_RRD=YES).

Layer names can be set and retrieved with the GDALSetDescription/GDALGetDescription calls on the Raster Band objects.

Some HFA band metadata exported to GDAL metadata:

- LAYER_TYPE - layer type (athematic, ...)
- OVERVIEWS_ALGORITHM - layer overviews algorithm ('IMAGINE 2X2 Resampling', 'IMAGINE 4X4 Resampling', and others)

6.64.3 Configuration Options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). Currently the following configuration options are supported by the HFA driver:

- **HFA_USE_RRD=[YES/NO]:** Whether to force creation of external overviews in Erdas rrd format and with .rrd file name extension (gdaladdo with combination -ro --config USE_RRD YES creates overview file with .aux extension).
- **HFA_COMPRESS_OVR=[YES/NO]:** Whether to create compressed overviews. Default is to only create compressed overviews when the file is compressed. This configuration option can be used when building external overviews for a base image that is not in Erdas Imagine format. Resulting overview file will use the rrd structure and have .aux extension.

  ```
  gdaladdo out.tif --config USE_RRD YES --config HFA_COMPRESS_OVR YES 2 4 8
  ```

  Erdas Imagine and older ArcGIS versions may recognize overviews for some image formats only if they have .rrd extension. In this case use:

  ```
  gdaladdo out.tif --config USE_RRD YES --config HFA_USE_RRD YES --config HFA_
  ↪COMPRESS_OVR YES 2 4 8
  ```

- **GDAL_HFA_OVR_BLOCKSIZE=value:** Defaults to `64`. The block size (tile width/height) used for overviews can be specified by setting this configuration option to a power-of-two value between 32 and 2048.
- **USE_SPILL=[YES/NO]:** Defaults to `NO`. Whether to use a spill file when creating a new overview.

6.64.4 See Also

- Implemented as hfadataset.cpp.
- More information, and other tools are available on the Imagine (.img) Reader page as saved by archive.org.
- Erdas.com

6.65 RST -- Idrisi Raster Format

Driver short name

RST

Driver built-in by default

This driver is built-in by default

This format is basically a raw one. There is just one band per files, except in the RGB24 data type where the Red, Green and Blue bands are store interleaved by pixels in the order Blue, Green and Red. The others data type are unsigned 8

bits integer with values from 0 to 255 or signed 16 bits integer with values from -32.768 to 32.767 or 32 bits single precision floating point.32 bits. The description of the file is stored in a accompanying text file, extension RDC.

The RDC image description file doesn't include color table, or detailed geographic referencing information. The color table if present can be obtained by another accompanying file using the same base name as the RST file and SMP as extension.

For geographical referencing identification, the RDC file contains information that points to a file that holds the geographic reference details. Those files uses extension REF and resides in the same folder as the RST image or more likely in the Idrisi installation folders.

Therefore the presence or absence of the Idrisi software in the running operation system will determine the way that this driver will work. By setting the environment variable IDRISIDIR pointing to the Idrisi main installation folder will enable GDAL to find more detailed information about geographical reference and projection in the REF files.

Note that the RST driver recognizes the name convention used in Idrisi for UTM and State Plane geographic reference so it doesn't need to access the REF files. That is the case for RDC file that specify "utm-30n" or "spc87ma1" in the "ref. system" field. Note that exporting to RST in any other geographical reference system will generate a suggested REF content in the comment section of the RDC file.

- ".rst" the raw image file
- ".rdc" the description file
- ".smp" the color table file
- ".ref" the geographical reference file

6.65.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.65.2 See Also

- Implemented as IdrisiDataset.cpp.
- www.idrisi.com

6.66 ILWIS -- Raster Map

Driver short name

ILWIS

Driver built-in by default

This driver is built-in by default

This driver implements reading and writing of ILWIS raster maps and map lists. Select the raster files with `the.mpr` (for raster map) or `the.mpl` (for maplist) extensions.

Features:

- Support for Byte, Int16, Int32 and Float64 pixel data types.
- Supports map lists with an associated set of ILWIS raster maps.
- Read and write geo-reference (.grf). Support for geo-referencing transform is limited to north-oriented GeoRef-Corner only. If possible the affine transform is computed from the corner coordinates.
- Read and write coordinate files (.csy). Support is limited to: Projection type of Projection and Lat/Lon type that are defined in .csy file, the rest of pre-defined projection types are ignored.

Limitations:

- Map lists with internal raster map storage (such as produced through Import General Raster) are not supported.
- ILWIS domain (.dom) and representation (.rpr) files are currently ignored.

NOTE: Implemented in ilwis.

See Also: http://www.itc.nl/ilwis/default.asp .

6.66.1 Driver capabilities

Supports CreateCopy()

This driver supports the *`GDALDriver::CreateCopy()`* operation

Supports Create()

This driver supports the *`GDALDriver::Create()`* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.67 IRIS -- Vaisala's weather radar software format

Driver short name

IRIS

Driver built-in by default

This driver is built-in by default

This read-only GDAL driver is designed to provide access to the products generated by the IRIS weather radar software.

IRIS software format includes a lot of products, and some of them aren't even raster. The driver can read currently:

- PPI (reflectivity and speed): Plan position indicator
- CAPPI: Constant Altitude Plan position indicator
- RAIN1: Hourly rainfall accumulation
- RAINN: N-Hour rainfall accumulation
- TOPS: Height for selectable dBZ contour
- VIL: Vertically integrated liquid for selected layer
- MAX: Column Max Z WF W/NS Sections

Most of the metadata is read.

Vaisala provides information about the format and software at http://www.vaisala.com/en/defense/products/weatherradar/Pages/IRIS.aspx.

NOTE: Implemented as irisdataset.cpp.

6.67.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.68 ISCE -- ISCE

Driver short name

ISCE

Driver built-in by default

This driver is built-in by default

Driver for the image formats used in the JPL's Interferometric synthetic aperture radar Scientific Computing Environment (ISCE). Only images with data types mappable to GDAL data types are supported.

Image properties are stored under the ISCE metadata domain, but there is currently no support to access underlying components elements and their properties. Likewise, ISCE domain metadata will be saved as properties in the image XML file.

Georeferencing is not yet implemented.

The ACCESS_MODE property is not currently honored.

The only creation option currently is SCHEME, which value (BIL, BIP, BSQ) determine the interleaving (default is BIP).

NOTE: Implemented as iscedataset.cpp.

6.68.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.69 ISG -- International Service for the Geoid

Added in version 3.1.

Driver short name

ISG

Driver built-in by default

This driver is built-in by default

Supports reading grids in the International Service for the Geoid text format, used for number of geoid models at http://www.isgeoid.polimi.it/Geoid/reg_list.html

Format specification is at https://www.isgeoid.polimi.it/Geoid/format_specs.html

NOTE: Implemented as aaigriddataset.cpp.

6.69.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Note

WGS84 will always be arbitrarily reported as the interpolation CRS of the grid. Consult grid documentation for exact CRS to apply.

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.70 ISIS2 -- USGS Astrogeology ISIS Cube (Version 2)

Driver short name

ISIS2

Driver built-in by default

This driver is built-in by default

ISIS2 is a format used by the USGS Planetary Cartography group to store and distribute planetary imagery data. GDAL provides read access to ISIS2 formatted imagery data.

ISIS2 files often have the extension .cub, sometimes with an associated .lbl (label) file. When a .lbl file exists it should be used as the dataset name rather than the .cub file.

In addition to support for most ISIS2 imagery configurations, this driver also reads georeferencing and coordinate system information as well as selected other header metadata.

Implementation of this driver was supported by the United States Geological Survey.

ISIS2 is part of a family of related formats including PDS and ISIS3.

6.70.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.70.2 See Also

- Implemented as isis2dataset.cpp.
- *PDS -- Planetary Data System v3* driver
- *ISIS3 -- USGS Astrogeology ISIS Cube (Version 3)* driver

6.71 ISIS3 -- USGS Astrogeology ISIS Cube (Version 3)

Driver short name

ISIS3

Driver built-in by default

This driver is built-in by default

ISIS3 is a format used by the USGS Planetary Cartography group to store and distribute planetary imagery data. GDAL provides read/creation/update access to ISIS3 formatted imagery data.

ISIS3 files often have the extension .cub, sometimes with an associated .lbl (label) file. When a .lbl file exists it should be used as the dataset name rather than the .cub file. The driver also supports imagery stored in a separate GeoTIFF file.

In addition to support for most ISIS3 imagery configurations, this driver also reads georeferencing and coordinate system information as well as selected other header metadata.

A mask band is attached to each source band. The value of this mask band is 0 when the pixel value is the NULL value or one of the low/high on-instrument/processed saturation value, or 255 when the pixel value is valid.

Implementation of this driver was supported by the United States Geological Survey.

ISIS3 is part of a family of related formats including PDS and ISIS2.

6.71.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.71.2 Metadata

The ISIS3 label can be retrieved as JSON-serialized content in the json:ISIS3 metadata domain.

For example:

```
$ python 3
from osgeo import gdal
ds = gdal.Open('../autotest/gdrivers/data/isis3_detached.lbl')
print(ds.GetMetadata_List('json:ISIS3')[0])
{
  "IsisCube":{
    "_type":"object",
    "Core":{
      "_type":"object",
      "StartByte":1,
      "^Core":"isis3_detached.cub",
      "Format":"BandSequential",
      "Dimensions":{
        "_type":"group",
        "Samples":317,
        "Lines":30,
        "Bands":1
      },
      "Pixels":{
        "_type":"group",
        "Type":"UnsignedByte",
        "ByteOrder":"Lsb",
        "Base":0.000000,
        "Multiplier":1.000000
      }
```

(continues on next page)

(continued from previous page)


```
    },
    "Instrument":{
      "_type":"group",
      "TargetName":"Mars"
    },
    "BandBin":{
      "_type":"group",
      "Center":1.000000,
      "OriginalBand":1
    },
    "Mapping":{
      "_type":"group",
      "ProjectionName":"Equirectangular",
      "CenterLongitude":184.412994,
      "TargetName":"Mars",
      "EquatorialRadius":{
        "value":3396190.000000,
        "unit":"meters"
      },
      "PolarRadius":{
        "value":3376200.000000,
        "unit":"meters"
      },
      "LatitudeType":"Planetographic",
      "LongitudeDirection":"PositiveWest",
      "LongitudeDomain":360,
      "MinimumLatitude":-14.822815,
      "MaximumLatitude":-14.727503,
      "MinimumLongitude":184.441132,
      "MaximumLongitude":184.496521,
      "UpperLeftCornerX":-4766.964984,
      "UpperLeftCornerY":-872623.628822,
      "PixelResolution":{
        "value":10.102500,
        "unit":"meters\/pixel"
      },
      "Scale":{
        "value":5864.945312,
        "unit":"pixels\/degree"
      },
      "CenterLatitude":-15.147000,
      "CenterLatitudeRadius":3394813.857978
    }
  },
  "Label":{
    "_type":"object",
    "Bytes":65536,
  },
  "History":{
    "_type":"object",
    "Name":"IsisCube",
    "StartByte":1,
```


(continues on next page)

(continued from previous page)

```
    "Bytes":957,
    "^History":"r0200357_10m_Jul20_o_i3_detatched.History.IsisCube",
    "_data": {
       "ASCII" : "[...snip...]"
    },
  },
  "OriginalLabel":{
    "_type":"object",
    "Name":"IsisCube",
    "StartByte":1,
    "Bytes":2482,
    "^OriginalLabel":"r0200357_10m_Jul20_o_i3_detatched.OriginalLabel.IsisCube",
    "_data": {
       "HEX" : "0102[...snip...]"
    },
  }
}
```

or

```
$ gdalinfo -json ../autotest/gdrivers/data/isis3_detached.lbl -mdd all
```

On creation, a source template label can be passed to the SetMetadata() interface in the "json:ISIS3" metadata domain.

6.71.3 Creation support

The ISIS3 driver supports updating imagery of existing datasets, creating new datasets through the CreateCopy() and Create() interfaces.

When using CreateCopy(), gdal_translate or gdalwarp, an effort is made to preserve as much as possible of the original label when doing ISIS3 to ISIS3 conversions. This can be disabled with the USE_SRC_LABEL=NO creation option.

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The available creation options are:

- **DATA_LOCATION=[LABEL/EXTERNAL/GEOTIFF.]:** Defaults to `LABEL`. To specify the location of pixel data. The default value is LABEL, ie imagery immediately follows the label. If using EXTERNAL, the imagery is put in a raw file whose filename is the main filename with a .cub extension. If using GEOTIFF, the imagery is put in a separate GeoTIFF file, whose filename is the main filename with a .tif extension.
- **GEOTIFF_AS_REGULAR_EXTERNAL=[YES/NO]:** Defaults to `YES`. Whether the GeoTIFF file, if uncompressed, should be registered as a regular raw file. Defaults to YES, so as to maximize the compatibility with earlier version of the ISIS3 driver.
- **GEOTIFF_OPTIONS=value:** Comma separated list of KEY=VALUE tuples to forward to the GeoTIFF driver. e.g. `GEOTIFF_OPTIONS=COMPRESS=LZW`.
- **EXTERNAL_FILENAME=[<filena/e>]:** Override default external filename. Only for DATA_LOCATION=EXTERNAL or GEOTIFF.
- **TILED=[YES/NO]:** Defaults to `NO`. Whether the pixel data should be tiled. Default is NO (ie band sequential organization).
- **BLOCKXSIZE=value:** Defaults to `256`. Tile width in pixels. Only used if *`TILED=YES`*.
- **BLOCKYSIZE=value:** Defaults to `256`. Tile height in pixels. Only used if *`TILED=YES`*.
- **COMMENT=value:** Comment to add into the label.

- **LATITUDE_TYPE=[Planetocentric/Planetographic]:** Defaults to `Planetocentric`. Value of Mapping.LatitudeType. If specified, and *USE_SRC_MAPPING* is in effect, this will be taken into account to override the source LatitudeType.
- **LONGITUDE_DIRECTION=[PositiveEast/PositiveWest]:** Defaults to `PositiveEast`. Value of Mapping.LongitudeDirection. If specified, and *USE_SRC_MAPPING* is in effect, this will be taken into account to override the source LongitudeDirection.
- **TARGET_NAME=value:** Value of Mapping.TargetName. This is normally deduced from the SRS datum name. If specified, and *USE_SRC_MAPPING* is in effect, this will be taken into account to override the source TargetName.
- **FORCE_360=[YES/NO]:** Defaults to `NO`. Whether to force longitudes in the [0, 360] range.
- **WRITE_BOUNDING_DEGREES=[YES/NO]:** Defaults to `YES`. Whether to write Min/MaximumLong/ Latitude values.
- **BOUNDING_DEGREES=<min_long,min_lat,max_long,max_lat>:** Manually set bounding box (values will not be modified by *LONGITUDE_DIRECTION* or *FORCE_360* options).
- **USE_SRC_LABEL=[YES/NO]:** Defaults to `YES`. Whether to use source label in ISIS3 to ISIS3 conversions.
- **USE_SRC_MAPPING=[YES/NO]:** Defaults to `NO`. Whether to use Mapping group from source label in ISIS3 to ISIS3 conversions. Defaults to NO (that is to say that the content of Mapping group will be created from new dataset geotransform and projection). Only used if *USE_SRC_LABEL=YES*
- **USE_SRC_HISTORY=[YES/NO]:** Defaults to `YES`. Whether to use the content pointed by the source History object in ISIS3 to ISIS3 conversions, and write it to the new dataset. Only used if *USE_SRC_LABEL=YES*. If *ADD_GDAL_HISTORY* and *USE_SRC_HISTORY* are set to YES (or unspecified), a new history section will be appended to the existing history.
- **ADD_GDAL_HISTORY=[YES/NO]:** Defaults to `YES`. Whether to add GDAL specific history in the content pointed by the History object in ISIS3 to ISIS3 conversions. Only used if *USE_SRC_LABEL=YES*. If *ADD_GDAL_HISTORY* and *USE_SRC_HISTORY* are set to YES (or unspecified), a new history section will be appended to the existing history. When *ADD_GDAL_HISTORY=YES*, the history is normally composed from current GDAL version, binary name and path, host name, user name and source and target filenames. It is possible to completely override it by specifying the *GDAL_HISTORY* option.
- **GDAL_HISTORY=value:** Manually defined GDAL history. Must be formatted as ISIS3 PDL. If not specified, it is automatically composed. Only used if *ADD_GDAL_HISTORY=YES* (or unspecified).

6.71.4 Open options

Added in version 3.12.

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The available open options are:

- **INCLUDE_OFFLINE_CONTENT=[YES/NO]:** Defaults to `YES`. Whether to include a `_data` member in `json:ISIS3` metadata with offline content of label objects. Defaults to YES
- **MAX_SIZE_OFFLINE_CONTENT=<integer>:** Defaults to `1000000000`. Maximum size of offline content to include in `_data` member, in bytes.

6.71.5 Examples

How to create a copy of a source ISIS3 dataset to another ISIS3 dataset while modifying a parameter of IsisCube.Mapping group, by using GDAL Python :

```
import json
from osgeo import gdal
```

(continues on next page)

(continued from previous page)

```
src_ds = gdal.Open('in.lbl')
# Load source label as JSON
label = json.loads( src_ds.GetMetadata_List('json:ISIS3')[0] )
# Update parameter
label["IsisCube"]["Mapping"]["TargetName"] = "Moon"

# Instantiate new raster
# Note the USE_SRC_MAPPING=YES creation option, since we modified the
# IsisCube.Mapping section, which otherwise is completely rewritten from
# the geotransform and projection attached to the output dataset.
out_ds = gdal.GetDriverByName('ISIS3').Create('out.lbl',
                                              src_ds.RasterXSize,
                                              src_ds.RasterYSize,
                                              src_ds.RasterCount,
                                              src_ds.GetRasterBand(1).DataType,
                                              options = ['USE_SRC_MAPPING=YES'])
# Attach the modified label
out_ds.SetMetadata( [json.dumps(label)], 'json:ISIS3' )

# Copy imagery (assumes that each band fits into memory, otherwise a line-by
# line or block-per-block strategy would be more appropriate )
for i in range(src_ds.RasterCount):
    out_ds.GetRasterBand(1).WriteRaster( 0, 0,
                                         src_ds.RasterXSize,
                                         src_ds.RasterYSize,
                                         src_ds.GetRasterBand(1).ReadRaster() )
out_ds = None
src_ds = None
```

6.71.6 See Also

- Implemented as isis3dataset.cpp.
- *GDAL PDS Driver*
- *GDAL ISIS2 Driver*

6.72 JDEM -- Japanese DEM (.mem)

Driver short name

JDEM

Driver built-in by default

This driver is built-in by default

GDAL includes read support for Japanese DEM files, normally having the extension .mem. These files are a product of the Japanese Geographic Survey Institute.

These files are represented as having one 32bit floating band with elevation data. The georeferencing of the files is returned as well as the coordinate system (always lat/long on the Tokyo datum).

There is no update or creation support for this format.

NOTE: Implemented as jdemdataset.cpp.

See Also: Geographic Survey Institute (GSI) Web Site.

6.72.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.73 JP2ECW -- ERDAS JPEG2000 (.jp2)

Driver short name

JP2ECW

Build dependencies

ECW SDK

GDAL supports reading and writing JPEG2000 files using the ERDAS ECW/JP2 SDK developed by Hexagon Geospatial (formerly Intergraph, ERDAS, ERMapper). Support is optional and requires linking in the libraries available from the ECW/JP2 SDK Download page.

Coordinate system and georeferencing transformations are read, and some degree of support is included for GeoJP2 (tm) (GeoTIFF-in-JPEG2000), ERDAS GML-in-JPEG2000, and the new GML-in-JPEG2000 specification developed at OGC.

6.73.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.73.2 Licensing

The ERDAS ECW/JP2 SDK v5.x is available under multiple license types. For Desktop usage, decoding any sized ECW/JP2 image is made available free of charge. To compress, deploy on a Server platform, or decode unlimited sized files on Mobile platforms a license must be purchased from Hexagon Geospatial.

6.73.3 History

- v3.x - Last release, 2006
- v4.x - Last release, 2012
- v5.x - Active development, 2013 - current

6.73.4 Georeferencing

Georeferencing information can come from different sources : internal (GeoJP2 or GMLJP2 boxes), worldfile .j2w/.wld sidecar files, or PAM (Persistent Auxiliary metadata) .aux.xml sidecar files. By default, information is fetched in following order (first listed is the highest priority): PAM, GeoJP2, GMLJP2, WORLDFILE.

The allowed sources and their priority order can be changed with the *GDAL_GEOREF_SOURCES* configuration option (or *GEOREF_SOURCES* open option) whose value is a comma-separated list of the following keywords : PAM, GEOJP2, GMLJP2, INTERNAL (shortcut for GEOJP2,GMLJP2), WORLDFILE, NONE. Earlier mentioned sources take priority over later ones. A non mentioned source will be ignored.

For example setting it to "WORLDFILE,PAM,INTERNAL" will make a geotransformation matrix from a potential worldfile priority over PAM or internal JP2 boxes. Setting it to "PAM,WORLDFILE,GEOJP2" will use the mentioned sources and ignore GMLJP2 boxes.

6.73.5 Open Options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are available:

- **1BIT_ALPHA_PROMOTION=[YES/NO]:** Defaults to `YES`. Whether a 1-bit alpha channel should be promoted to 8-bit.
- **GEOREF_SOURCES=value:** Define which georeferencing sources are allowed and their priority order. See *Georeferencing* paragraph.

6.73.6 Creation Options:

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). Supported creation options are listed below.

Note: Only Licensing and compression target need to be specified. The ECW/JP2 SDK will default all other options to recommended settings based on the input characteristics. Changing other options can *substantially* impact decoding speed and compatibility with other JPEG2000 toolkits.

- **LARGE_OK=YES:** (v3.x SDK only) Allow compressing files larger than 500MB in accordance with EULA terms. Deprecated since v4.x and replaced by *ECW_ENCODE_KEY* & *ECW_ENCODE_COMPANY*.
- **ECW_ENCODE_KEY=<key>:** (v4.x SDK or higher) Provide the OEM encoding key to unlock encoding capability up to the licensed gigapixel limit. The key is approximately 129 hex digits long. The Company and Key must match and must be re-generated with each minor release of the SDK. It may also be provided globally as a configuration option.
- **ECW_ENCODE_COMPANY=<name>:** *(v4.x SDK or higher)* Provide the name of the company in the issued OEM key (see ECW_ENCODE_KEY). The Company and Key must match and must be re-generated with each minor release of the SDK. It may also be provided globally as a configuration option.
- **TARGET=<percent>:** Defaults to `75`. Set the target size reduction as a percentage of the original. If not provided defaults to 75 for an 75% reduction. TARGET=0 uses lossless compression.
- **PROJ=value:** Name of the ECW projection string to use. Common examples are NUTM11, or GEODETIC.
- **DATUM=value:** Name of the ECW datum string to use. Common examples are WGS84 or NAD83.
- **GMLJP2=[YES/NO]:** Defaults to `YES`. Indicates whether a GML box conforming to the OGC GML in JPEG2000 specification should be included in the file. Unless GMLJP2V2_DEF is used, the version of the GMLJP2 box will be version 1.
- **GMLJP2V2_DEF=[<filename>/<json>/YES]:** Indicates whether a GML box conforming to the OGC GML in JPEG2000, version 2 specification should be included in the file. *filename* must point to a file with a JSON content that defines how the GMLJP2 v2 box should be built. See *GMLJP2v2 definition file section* in documentation of the JP2OpenJPEG driver for the syntax of the JSON configuration file. It is also possible to directly pass the JSON content inlined as a string. If filename is just set to YES, a minimal instance will be built.
- **GeoJP2=[YES/NO]:** Defaults to `YES`. Indicates whether a UUID/GeoTIFF box conforming to the GeoJP2 (GeoTIFF in JPEG2000) specification should be included in the file.
- **PROFILE=[BASELINE_0/BASELINE_1/BASELINE_2/NPJE/EPJE]:** Review the ECW SDK documentation for details on profile meanings.
- **PROGRESSION=[LRCP/RLCP/RPCL]:** Defaults to `RPCL`. Set the progression order with which the JPEG2000 codestream is written.
- **CODESTREAM_ONLY=[YES/NO]:** Defaults to `NO`. If set to YES, only the compressed imagery code stream will be written. If NO a JP2 package will be written around the code stream including a variety of meta information.
- **LEVELS=<integer>:** Resolution levels in pyramid (by default so many that the size of the smallest thumbnail image is 64x64 pixels at maximum)
- **LAYERS=value:** Defaults to `1`. Quality layers (default, 1)
- **PRECINCT_WIDTH=value:** Defaults to `64`. Precinct Width
- **PRECINCT_HEIGHT=value:** Defaults to `64`. Precinct Height
- **TILE_WIDTH=value:** Tile Width (default, image width eg. 1 tile). Apart from GeoTIFF, in JPEG2000 tiling is not critical for speed if precincts are used. The minimum tile size allowed by the standard is 1024x1024 pixels.
- **TILE_HEIGHT=value:** Tile Height (default, image height eg. 1 tile)
- **INCLUDE_SOP=[YES/NO]:** Output Start of Packet Marker (default false)
- **INCLUDE_EPH=[YES/NO]:** Output End of Packet Header Marker (default true)

- **DECOMPRESS_LAYERS=value:** The number of quality layers to decode
- **DECOMPRESS_RECONSTRUCTION_PARAMETER=[IRREVERSIBLE_9x7/REVERSIBLE_5x3]:**
- **WRITE_METADATA=[YES/NO]:** Whether metadata should be written, in a dedicated JP2 XML box. Defaults to NO. The content of the XML box will be like:

```
<GDALMultiDomainMetadata>
  <Metadata>
    <MDI key="foo">bar</MDI>
  </Metadata>
  <Metadata domain='aux_domain'>
    <MDI key="foo">bar</MDI>
  </Metadata>
  <Metadata domain='a_xml_domain' format='xml'>
    <arbitrary_xml_content>
    </arbitrary_xml_content>
  </Metadata>
</GDALMultiDomainMetadata>
```

  If there are metadata domain whose name starts with "xml:BOX_", they will be written each as separate JP2 XML box.

  If there is a metadata domain whose name is "xml:XMP", its content will be written as a JP2 UUID XMP box.

- **MAIN_MD_DOMAIN_ONLY=[YES/NO]:** Defaults to `NO`. (Only if *`WRITE_METADATA=YES`*) Whether only metadata from the main domain should be written.

"JPEG2000 format does not support creation of GDAL overviews since the format is already considered to be optimized for "arbitrary overviews". JP2ECW driver also arranges JP2 codestream to allow optimal access to power of two overviews. This is controlled with the creation option LEVELS."

6.73.7 Create support

While the driver advertises the Create() capability, contrary to most other drivers that implement it, the implementation of RasterIO() and WriteBlock() in the JP2ECW driver does not support arbitrary random writing. Data must be written in the dataset from top to bottom, whole line(s) at a time.

6.73.8 Configuration Options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The ERDAS ECW/JP2 SDK supports a variety of runtime configuration options to control various features. See the ECW/JP2 SDK documentation for full details on the meaning of these options.

- **ECW_CACHE_MAXMEM=<bytes>:** maximum bytes of RAM used for in-memory caching. If not set, up to one quarter of physical RAM will be used by the SDK for in-memory caching.
- **ECW_TEXTURE_DITHER=[TRUE/FALSE]:** Defaults to `TRUE`. This may be set to FALSE to disable dithering when decompressing ECW files.
- **ECW_FORCE_FILE_REOPEN=[TRUE/FALSE]:** Defaults to `FALSE`. This may be set to TRUE to force open a file handle for each file for each connection made.
- **ECW_CACHE_MAXOPEN=value:** Defaults to `unlimited`. The maximum number of files to keep open for ECW file handle caching.

- **ECW_AUTOGEN_J2I=[TRUE/FALSE]:** Defaults to TRUE. Controls whether .j2i index files should be created when opening jpeg2000 files.
- **ECW_RESILIENT_DECODING=[TRUE/FALSE]:** Defaults to TRUE. Controls whether the reader should be forgiving of errors in a file, trying to return as much data as is available. If set to FALSE an invalid file will result in an error.

6.73.9 Metadata

XMP metadata can be extracted from JPEG2000 files, and will be stored as XML raw content in the xml:XMP metadata domain.

ECW/JP2 SDK v5.1+ also advertises JPEG2000 structural information as generic File Metadata reported under "JPEG2000" metadata domain (-mdd):

- **ALL_COMMENTS**: Generic comment text field
- **PROFILE**: Profile type (0,1,2). Refer to ECW/JP2 SDK documentation for more info
- **TILES_X**: Number of tiles on X (horizontal) Axis
- **TILES_Y**: Number of tiles on Y (vertical) Axis
- **TILE_WIDTH**: Tile size on X Axis
- **TILE_HEIGHT**: Tile size on Y Axis
- **PRECINCT_SIZE_X**: Precinct size for each resolution level (smallest to largest) on X Axis
- **PRECINCT_SIZE_Y**: Precinct size for each resolution level (smallest to largest) on Y Axis
- **CODE_BLOCK_SIZE_X**: Code block size on X Axis
- **CODE_BLOCK_SIZE_Y**: Code block size on Y Axis
- **PRECISION**: Precision / Bit-depth of each component eg. 8,8,8 for 8bit 3 band imagery.
- **RESOLUTION_LEVELS**: Number of resolution levels
- **QUALITY_LAYERS**: Number of quality layers
- **PROGRESSION_ORDER**: Progression order (RPCL, LRCP, CPRL, RLCP)
- **TRANSFORMATION_TYPE**: Filter transformation used (9x7, 5x3)
- **USE_SOP**: Start of Packet marker detected (TRUE/FALSE)
- **USE_EPH**: End of Packet header marker detected (TRUE/FALSE)
- **GML_JP2_DATA**: OGC GML GeoReferencing box detected (TRUE/FALSE)
- **COMPRESSION_RATE_TARGET**: Target compression rate used on encoding

6.73.10 See Also

- Implemented as ecwdataset.cpp.
- ECW/JP2 SDK available at www.hexagongeospatial.com
- Further product information available in the User Guide
- Support for non-GDAL specific issues should be directed to the Hexagon Geospatial public forum

6.74 JP2Grok -- JPEG2000 driver based on Grok library

Driver short name

JP2Grok

Build dependencies

Grok >= 20.2

Added in version 3.13.

This driver is an implementation of a JPEG2000 reader/writer based on the Grok library.

The driver uses the VSI Virtual File API, so it can read JPEG2000 compressed NITF files. It also supports native S3 access through Grok's built-in cloud I/O.

XMP metadata can be extracted from JPEG2000 files, and will be stored as XML raw content in the xml:XMP metadata domain.

The driver supports writing georeferencing information as GeoJP2 and GMLJP2 boxes.

The driver supports creating files with transparency, arbitrary band count, and adding/reading metadata. Update of georeferencing or metadata of existing file is also supported. Optional intellectual property metadata can be read/written in the xml:IPR box.

6.74.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.74.2 Georeferencing

Georeferencing information can come from different sources : internal (GeoJP2 or GMLJP2 boxes), worldfile .j2w/.wld sidecar files, or PAM (Persistent Auxiliary metadata) .aux.xml sidecar files. By default, information is fetched in following order (first listed is the highest priority): PAM, GeoJP2, GMLJP2, WORLDFILE.

The allowed sources and their priority order can be changed with the `GDAL_GEOREF_SOURCES` configuration option (or `GEOREF_SOURCES` open option) whose value is a comma-separated list of the following keywords : PAM, GEOJP2, GMLJP2, INTERNAL (shortcut for GEOJP2,GMLJP2), WORLDFILE, NONE. Earlier mentioned sources take priority over later ones. A non mentioned source will be ignored.

For example setting it to "WORLDFILE,PAM,INTERNAL" will make a geotransformation matrix from a potential worldfile priority over PAM or internal JP2 boxes. Setting it to "PAM,WORLDFILE,GEOJP2" will use the mentioned sources and ignore GMLJP2 boxes.

6.74.3 Thread support

The Grok library uses multi-threaded decompression internally via its own thread pool. Multi-threaded encoding is also supported. The number of threads is determined at runtime from GDALGetNumThreads.

When a JPEG2000 file contains multiple tiles, GDAL will additionally dispatch tile processing across threads for parallel CopyTileData operations during decoding.

6.74.4 Open Options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are available:

- **1BIT_ALPHA_PROMOTION=[YES/NO]:** Defaults to `YES`. Whether a 1-bit alpha channel should be promoted to 8-bit.
- **GEOREF_SOURCES=value:** Define which georeferencing sources are allowed and their priority order. See *Georeferencing* paragraph.
- **USE_TILE_AS_BLOCK=[YES/NO]:** Defaults to `NO`. Whether to always use the JPEG-2000 block size as the GDAL block size. Setting this option can be useful when doing whole image decompression and the image is single-tiled. Note however that the tile size must not exceed 2 GB since that's the limit supported by GDAL.

6.74.5 Creation Options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following creation options are available:

- **CODEC=[JP2/J2K]:** JP2 will add JP2 boxes around the codestream data. The value is determined automatically from the file extension. If it is neither JP2 nor J2K, J2K codec is used.
- **GMLJP2=[YES/NO]:** Defaults to `YES`. Indicates whether a GML box conforming to the OGC GML in JPEG2000 specification should be included in the file. Unless GMLJP2V2_DEF is used, the version of the GMLJP2 box will be version 1.
- **GMLJP2V2_DEF=[YES/<filename>/<json>]:** Indicates whether a GML box conforming to the OGC GML in JPEG2000, version 2.0.1 specification should be included in the file. *filename* must point to a file with a JSON content that defines how the GMLJP2 v2 box should be built. See the *JP2OpenJPEG -- JPEG2000 driver based on OpenJPEG library* driver documentation for the syntax of the JSON configuration file. It is also possible to directly pass the JSON content inlined as a string. If filename is just set to YES, a minimal instance will be built.
- **GeoJP2=[YES/NO]:** Defaults to `YES`. Indicates whether a UUID/GeoTIFF box conforming to the GeoJP2 (GeoTIFF in JPEG2000) specification should be included in the file.
- **QUALITY=<float_value,float_value,...>:** Percentage between 0 and 100. A value of 50 means the file will be half-size in comparison to uncompressed data, 33 means 1/3, etc.. Defaults to 25 (unless the dataset is made of a single band with color table, in which case the default quality is 100). It is possible to specify several quality values (comma separated) to ask for several quality layers. Quality values should be increasing.
- **REVERSIBLE=[YES/NO]:** YES means use of reversible 5x3 integer-only filter, NO use of the irreversible DWT 9-7. Defaults to NO (unless the dataset is made of a single band with color table, in which case reversible filter is used).
- **RESOLUTIONS=<integer>:** Number of resolution levels. Default value is selected such the smallest overview of a tile is no bigger than 128x128.

- **BLOCKXSIZE=<integer>:** Defaults to `1024`. Tile width.
- **BLOCKYSIZE=<integer>:** Defaults to `1024`. Tile height.
- **PROGRESSION=[LRCP/RLCP/RPCL/PCRL/CPRL]:** Defaults to `LRCP`. Progression order.
- **SOP=[YES/NO]:** Defaults to `NO`. YES means generate SOP (Start Of Packet) marker segments.
- **EPH=[YES/NO]:** Defaults to `NO`. YES means generate EPH (End of Packet Header) marker segments.
- **YCBCR420=[YES/NO]:** Defaults to `NO`. YES if RGB must be resampled to YCbCr 4:2:0.
- **YCC=[YES/NO]:** Defaults to `YES`. YES if RGB must be transformed to YCC color space ("MCT transform", i.e. internal transform, without visual degradation).
- **NBITS=<integer>:** Bits (precision) for sub-byte files (1-7), sub-uint16 (9-15), sub-uint32 (17-31).
- **1BIT_ALPHA=[YES/NO]:** Whether to encode the alpha channel as a 1-bit channel (when there's an alpha channel). Defaults to NO, unless *INSPIRE_TG=YES*. Enabling this option might cause compatibility problems with some readers.
- **ALPHA=[YES/NO]:** Defaults to `NO`. Whether to force encoding last channel as alpha channel. Only useful if the color interpretation of that channel is not already Alpha.
- **PROFILE=[AUTO/UNRESTRICTED/PROFILE_1]:** Defaults to `AUTO`. Determine which codestream profile to use. UNRESTRICTED corresponds to the "Unrestricted JPEG 2000 Part 1 codestream" (RSIZ=0). PROFILE_1 corresponds to the "JPEG 2000 Part 1 Profile 1 codestream" (RSIZ=2), which add constraints on tile dimensions and number of resolutions. In AUTO mode, the driver will determine if the *BLOCKXSIZE*, *BLOCKYSIZE*, *RESOLUTIONS*, *CODEBLOCK_WIDTH* and *CODEBLOCK_HEIGHT* values are compatible with PROFILE_1 and advertise it in the relevant case. Note that the default values of those options are compatible with PROFILE_1. Otherwise UNRESTRICTED is advertised.
- **INSPIRE_TG=[YES/NO]:** Defaults to `NO`. Whether to use JPEG2000 features that comply with Inspire Orthoimagery Technical Guidelines. When set to YES, implies *PROFILE=PROFILE_1*, *1BIT_ALPHA=YES*, *GEOBOXES_AFTER_JP2C=YES*. The *CODEC*, *BLOCKXSIZE*, *BLOCKYSIZE*, *RESOLUTIONS*, *NBITS*, *PROFILE*, *CODEBLOCK_WIDTH* and *CODEBLOCK_HEIGHT* options will be checked against the requirements and recommendations of the Technical Guidelines.
- **JPX=[YES/NO]:** Defaults to `YES`. Whether to advertise JPX features, and add a Reader requirement box, when a GMLJP2 box is written. This option should not be used unless compatibility problems with a reader occur.
- **GEOBOXES_AFTER_JP2C=[YES/NO]:** Whether to place GeoJP2/GMLJP2 boxes after the code-stream. Defaults to NO, unless *INSPIRE_TG=YES*. This option should not be used unless compatibility problems with a reader occur.
- **PRECINCTS=[<{prec_w,prec_h},{prec_w,prec_h}/...>]:** A list of {precincts width,precincts height} tuples to specify precincts size. Each value should be a multiple of 2. The maximum number of tuples used will be the number of resolutions. The first tuple corresponds to the higher resolution level, and the following ones to the lower resolution levels. If less tuples are specified, the last one is used by dividing its values by 2 for each extra lower resolution level. The default value used is {512,512},{256,512},{128,512},{64,512},{32,512},{16,512},{8,512},{4,512},{2,512}. An empty string may be used to disable precincts (i.e. the default {32767,32767},{32767,32767}, ... will then be used).
- **TILEPARTS=[DISABLED/RESOLUTIONS/LAYERS/COMPONENTS]:** Defaults to `DISABLED`. Whether to generate tile-parts and according to which criterion.
- **CODEBLOCK_WIDTH=<integer>:** Defaults to `64`. Codeblock width: power of two value between 4 and 1024. Note that CODEBLOCK_WIDTH * CODEBLOCK_HEIGHT must not be greater than 4096. For PROFILE_1 compatibility, CODEBLOCK_WIDTH must not be greater than 64.

- **CODEBLOCK_HEIGHT=<integer>:** Defaults to 64. Codeblock height: power of two value between 4 and 1024. Note that CODEBLOCK_WIDTH * CODEBLOCK_HEIGHT must not be greater than 4096. For PROFILE_1 compatibility, CODEBLOCK_HEIGHT must not be greater than 64.
- **CODEBLOCK_STYLE=value:** Style of the code-block coding passes. The following 6 independent settings can be combined together (values should be comma separated):
  - *BYPASS* (1): enable selective arithmetic coding bypass (can substantially improve coding/decoding speed, at the expense of larger file size)
  - *RESET* (2): reset context probabilities on coding pass boundaries
  - *TERMALL* (4): enable termination on each coding pass
  - *VSC* (8): enable vertically causal context
  - *PREDICTABLE* (16): enable predictable termination (helps for error detection)
  - *SEGSYM* (32): enable segmentation symbols (helps for error detection)

  Instead of specifying them by text, it is also possible to give the corresponding numeric value of the global codeblock style, by adding the selected options (for example "BYPASS,TERMALL" is equivalent to "5"=1+4)

  By default, none of them are enabled. Enabling them will generally increase codestream size, but improve either coding/decoding speed or resilience/error detection.
- **PLT=[YES/NO]:** Defaults to NO. Whether to write a PLT (Packet Length) marker segment in tile-part headers.
- **TLM=[YES/NO]:** Defaults to NO. Whether to write a TLM (Tile-part Length) marker segment in main header.
- **WRITE_METADATA=[YES/NO]:** Defaults to NO. Whether metadata should be written, in a dedicated JP2 'xml ' box. The content of the 'xml ' box will be like:

```
<GDALMultiDomainMetadata>
  <Metadata>
    <MDI key="foo">bar</MDI>
  </Metadata>
  <Metadata domain='aux_domain'>
    <MDI key="foo">bar</MDI>
  </Metadata>
  <Metadata domain='a_xml_domain' format='xml'>
    <arbitrary_xml_content>
    </arbitrary_xml_content>
  </Metadata>
</GDALMultiDomainMetadata>
```

  If there are metadata domain whose name starts with "xml:BOX_", they will be written each as separate JP2 'xml ' box.

  If there is a metadata domain whose name is "xml:XMP", its content will be written as a JP2 'uuid' XMP box.

  If there is a metadata domain whose name is "xml:IPR", its content will be written as a JP2 'jp2i' box.
- **MAIN_MD_DOMAIN_ONLY=[YES/NO]:** Defaults to NO. (Only if *WRITE_METADATA=YES*) Whether only metadata from the main domain should be written.
- **USE_SRC_CODESTREAM=[YES/NO]:** (EXPERIMENTAL!) When source dataset is JPEG2000, whether to reuse the codestream of the source dataset unmodified. Defaults to NO. Note that enabling that feature might result in inconsistent content of the JP2 boxes w.r.t. to the content of the source codestream. Most other creation options will be ignored in that mode. Can be useful in some use cases when adding/correcting georeferencing, metadata, ...

- **TRANSCODE=[YES/NO]:** Defaults to NO. When the source dataset is JPEG 2000, transcode the codestream without a full decompression/recompression round-trip. The underlying codestream data is copied verbatim while the following options may be used to modify markers or progression:
  - *PLT* – insert PLT (Packet Length, Tile-part) marker segments
  - *TLM* – insert TLM (Tile-part Length) marker segments
  - *SOP* – insert SOP (Start Of Packet) marker segments
  - *EPH* – insert EPH (End of Packet Header) marker segments
  - *PROGRESSION* – change the progression order (LRCP, RLCP, RPCL, PCRL, CPRL)

  JP2 metadata boxes (GeoTIFF UUID, XMP, GMLJP2, IPR, GDAL xml) are regenerated from the current dataset metadata.

  Because the codestream coefficients are copied rather than re-encoded, creation options that would affect the pixel data (QUALITY, REVERSIBLE, BLOCKXSIZE, BLOCKYSIZE, RESOLUTIONS, YCBCR420, YCC, NBITS, 1BIT_ALPHA, PRECINCTS, TILEPARTS, CODEBLOCK_WIDTH, CODEBLOCK_HEIGHT, CODEBLOCK_STYLE, USE_SRC_CODESTREAM) are ignored — a warning is emitted if any are supplied.

  Example:

```
gdal_translate -of JP2Grok -co TRANSCODE=YES \
    -co PLT=YES -co TLM=YES -co PROGRESSION=RPCL \
    input.jp2 output.jp2
```

- **COMMENT=<string>:** Content of the comment (COM) marker in the codestream.

6.74.5.1 Lossless compression

Lossless compression can be achieved if ALL the following creation options are defined :

- *QUALITY=100*
- *REVERSIBLE=YES*
- *YCBCR420=NO* (which is the default)

6.74.6 Standalone plugin compilation

While this driver may be built as part of a whole GDAL build, either in libgdal itself, or as a plugin, it is also possible to only build this driver as a plugin, against an already built libgdal.

The version of the GDAL sources used to build the driver must match the version of the libgdal it is built against.

For example, from a "build_grok" directory under the root of the GDAL source tree:

```
cmake -S ../frmts/grok -DCMAKE_PREFIX_PATH=/path/to/GDAL_installation_prefix -DGROK_
↪ROOT=/path/to/grok_install_prefix
cmake --build .
```

Note that such a plugin, when used against a libgdal not aware of it, will be systematically loaded at GDAL driver initialization time, and will not benefit from *deferred plugin loading capabilities <rfc-96>*. For that, libgdal itself must be built with the CMake variable GDAL_REGISTER_DRIVER_JP2GROK_FOR_LATER_PLUGIN=ON set.

6.74.7 See Also

- Implemented as grokdataset.cpp.
- The Grok library home page

Other JPEG2000 GDAL drivers :

- *JP2OpenJPEG: based on OpenJPEG library (open source)*
- *JP2ECW: based on Erdas ECW library (proprietary)*
- *JP2MRSID: based on LizardTech MrSID library (proprietary)*
- *JP2KAK: based on Kakadu library (proprietary)*

6.75 JP2KAK -- JPEG 2000 (based on Kakadu SDK)

Driver short name

JP2KAK

Build dependencies

Kakadu SDK

The JP2KAK driver, which uses the proprietary Kakadu SDK, supports JPEG 2000 images, which are specified in the Rec. ITU-T T.8xx | ISO/IEC 15444 family of standards. JPEG 2000 uses a substantially different format and compression mechanism than the traditional JPEG compression and JPEG JFIF format. JPEG and JPEG 2000 are distinct compression standards produced by the same group. JPEG 2000 is based on wavelet compression.

The driver includes support for:

- reading both JPEG 2000 codestreams (.j2c or .jpc) and JP2 files (.jp2). New images can be also written, but existing images cannot be updated in place.
- lossy and lossless compression of 8-bit and 16-bit images with 1 or more bands (components).
- GeoTIFF style coordinate system and georeferencing information in JP2 file via the GeoJP2(tm) mechanism.
- JPEG 2000 Part 1 (Rec. ITU-T T.800 | ISO/IEC 15444-1), Part 2 (Rec. ITU-T T.801 | ISO/IEC 15444-2) and Part 15 (HTJ2K) (Rec. ITU-T T.814 | ISO/IEC 15444-15)

The Kakadu SDK is a high quality and high performance JPEG 2000 library in wide used in the geospatial and general imaging community. However, it is not free, and, by default, a builds of GDAL from source does not include support for the JP2KAK driver. The builder must acquire a license for the Kakadu SDK and configure GDAL accordingly.

When reading images this driver will represent the bands as being Byte (8-bit unsigned), 16-bit signed/unsigned, and 32-bit signed/unsigned. Georeferencing and coordinate system information will be available if the file is a GeoJP2 (tm) file. Files color encoded in YCbCr color space will be automatically translated to RGB. Paletted images are also supported.

XMP metadata can be extracted from JPEG 2000 files, and will be stored as XML raw content in the xml:XMP metadata domain.

6.75.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.75.2 Configuration Options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). The JP2KAK driver supports the following configuration options:

- **JP2KAK_THREADS=value:** By default an effort is made to take advantage of multi-threading on multi-core computers using default rules from the Kakadu SDK. This option may be set to a value of zero to avoid using additional threads or to a specific count to create the requested number of worker threads.
- **JP2KAK_FUSSY=[YES/NO]:** Defaults to `NO`. This can be set to YES to turn on fussy reporting of problems with the JPEG 2000 data stream.
- **JP2KAK_RESILIENT=[YES/NO]:** Defaults to `NO`. This can be set to YES to force Kakadu to maximize resilience with incorrectly created JPEG 2000 data files, likely at some cost in performance. This is likely to be necessary if, among other reasons, you get an error message about "Expected to find EPH marker following packet header" or error reports indicating the need to run with the resilient and sequential flags on.
- **USE_TILE_AS_BLOCK=[YES/NO]:** Defaults to `NO`. Whether to use the JPEG 2000 block size as the GDAL block size.

6.75.3 Georeferencing

Georeferencing information can come from different sources : internal (GeoJP2 or GMLJP2 boxes), worldfile .j2w/.wld sidecar files, or PAM (Persistent Auxiliary metadata) .aux.xml sidecar files. By default, information is fetched in following order (first listed is the highest priority): PAM, GeoJP2, GMLJP2, WORLDFILE.

The allowed sources and their priority order can be changed with the *GDAL_GEOREF_SOURCES* configuration option (or *GEOREF_SOURCES* open option) whose value is a comma-separated list of the following keywords : PAM, GEOJP2, GMLJP2, INTERNAL (shortcut for GEOJP2,GMLJP2), WORLDFILE, NONE. Earlier mentioned sources take priority over later ones. A non mentioned source will be ignored.

For example setting it to "WORLDFILE,PAM,INTERNAL" will make a geotransformation matrix from a potential worldfile priority over PAM or internal JP2 boxes. Setting it to "PAM,WORLDFILE,GEOJP2" will use the mentioned sources and ignore GMLJP2 boxes.

6.75.4 Option Options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are available:

- **1BIT_ALPHA_PROMOTION=[YES/NO]:** Defaults to `YES`. Whether a 1-bit alpha channel should be promoted to 8-bit.
- **GEOREF_SOURCES=value:** Define which georeferencing sources are allowed and their priority order. See *Georeferencing* paragraph.

6.75.5 Creation Issues

JPEG 2000 files can only be created using the CreateCopy mechanism to copy from an existing dataset.

JPEG 2000 overviews are maintained as part of the mathematical description of the image. Overviews cannot be built as a separate process, but on read the image will generally be represented as having overview levels at various power of two factors.

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following creation options are supported:

- **CODEC=[JP2/J2K]:** Codec to use. If not specified, guess based on file extension. If unknown, default to JP2.
- **QUALITY=value:** Defaults to `20`. Set the compressed size ratio as a percentage of the size of the uncompressed image. The default is 20 indicating that the resulting image should be 20% of the size of the uncompressed image. Actual final image size may not exactly match that requested depending on various factors. A value of 100 will result in use of the lossless compression algorithm . On typical image data, if you specify a value greater than 65, it might be worth trying with *QUALITY=100* instead as lossless compression might produce better compression than lossy compression.
- **BLOCKXSIZE=value:** Defaults to `20000`. Set the tile width to use.
- **BLOCKYSIZE=value:** Set the tile height to use. Defaults to image height.
- **FLUSH=[TRUE/FALSE]:** Defaults to `TRUE`. Enable/Disable incremental flushing when writing files. Required to be FALSE for RLPC and LRPC Corder. May use a lot of memory when FALSE while writing large images.
- **GMLJP2=[YES/NO]:** Defaults to `YES`. Indicates whether a GML box conforming to the OGC GML in JPEG 2000 specification should be included in the file. Unless GMLJP2V2_DEF is used, the version of the GMLJP2 box will be version 1.
- **GMLJP2V2_DEF=[<filename>/<json>/YES]:** Indicates whether a GML box conforming to the OGC GML in JPEG 2000, version 2 specification should be included in the file. *filename* must point to a file with a JSON content that defines how the GMLJP2 v2 box should be built. See *GMLJP2v2 definition file section* in documentation of the JP2OpenJPEG driver for the syntax of the JSON configuration file. It is also possible to directly pass the JSON content inlined as a string. If filename is just set to YES, a minimal instance will be built.
- **GeoJP2=[YES/NO]:** Defaults to `YES`. Indicates whether a UUID/GeoTIFF box conforming to the GeoJP2 (GeoTIFF in JPEG 2000) specification should be included in the file.
- **LAYERS=value:** Defaults to `12`. Control the number of layers produced. These are sort of like resolution layers, but not exactly. The default value of 12 works well in most situations.
- **ROI=<xoff,yoff,xsize,ysize>:** Selects a region to be a region of interest to process with higher data quality. The various "R" flags below may be used to control the amount better. For example the settings "ROI=0,0,100,100", "Rweight=7" would encode the top left 100x100 area of the image with considerable higher quality compared to the rest of the image.

The following creation options are tightly tied to the Kakadu SDK, and are considered to be for advanced use only. Consult the Kakadu SDK documentation to better understand their meaning.

- **Corder**: Progression order. Defaults to "PRCL".
- **Cprecincts**: Precincts settings. Defaults to "{512,512},{256,512},{128,512},{64,512},{32,512},{16,512},{8,512},{4,512},{2,51
- **ORGgen_plt**: Whether to generate PLT markers (Packet Length). Defaults to "yes".
- **ORGgen_tlm**: Whether to generate TLM markers (Tile Length). Kakadu SDK defaults used.
- **ORGtparts**: Kakadu SDK defaults used.
- **Cmodes**: Kakadu SDK defaults used.
- **Clevels**: Quality levels. Kakadu SDK defaults used.
- **Cycc** =YES/NO: Whether to use YCbCr (lossy compression) or YUV (lossless compression) color space when encoding RGB images. Default to YES starting with GDAL 3.10 (was hardcoded to NO in previous versions)
- **Rshift**: Kakadu SDK defaults used.
- **Rlevels**: Kakadu SDK defaults used.
- **Rweight**: Kakadu SDK defaults used.
- **Qguard**: Kakadu SDK defaults used.
- **Creversible** = YES/NO: If not set and QUALITY >= 99.5, set to "yes", otherwise to "false".
- **Sprofile**: Kakadu SDK defaults used.
- **RATE**: Kakadu SDK defaults used. One or more bit-rates, expressed in terms of the ratio between the total number of compressed bits (including headers) and the product of the largest horizontal and vertical image component dimensions. A dash, -, may be used in place of the first bit-rate in the list to indicate that the final quality layer should include all compressed bits. If Clayers is not used, the number of layers is set to the number of rates specified here. If Clayers is used to specify an actual number of quality layers, one of the following must be true: 1) the number of rates specified here is identical to the specified number of layers; or 2) one or two rates are specified using this argument. When two rates are specified, the number of layers must be 2 or more and intervening layers will be assigned roughly logarithmically spaced bit-rates. When only one rate is specified, an internal heuristic determines a lower bound and logarithmically spaces the layer rates over the range. The rates have to be in ASC order.

6.75.6 Known Kakadu SDK Issues

6.75.6.1 Alpha Channel Writing in v7.8

Kakadu SDK v7.8 has a bug in jp2_channels::set_opacity_mapping that can cause an error when writing images with an alpha channel. Please upgrade to version 7.9.

```
Error: GdalIO: Error in Kakadu File Format Support: Attempting to
create a Component Mapping (cmap) box, one of whose channels refers to
a non-existent image component or palette lookup table. (code = 1)
```

6.75.6.2 kdu_get_num_processors always returns 0 for some platforms

On non-windows / non-mac installs (e.g. Linux), Kakadu SDK might not include unistd.h in kdu_arch.cpp. This means that _SC_NPROCESSORS_ONLN and _SC_NPROCESSORS_CONF are not defined and kdu_get_num_processors will always return 0. Therefore the jp2kak driver might not default to creating worker threads.

6.75.7 Standalone plugin compilation

Added in version 3.10.

While this driver may be built as part of a whole GDAL build, either in libgdal itself, or as a plugin, it is also possible to only build this driver as a plugin, against an already built libgdal.

The version of the GDAL sources used to build the driver must match the version of the libgdal it is built against.

For example, from a "build_jp2kak" directory under the root of the GDAL source tree:

```
cmake -S ../frmts/jp2kak -DCMAKE_PREFIX_PATH=/path/to/GDAL_installation_prefix -DKDU_
↪ROOT=/path/to/kakadu_root
cmake --build .
```

Note that such a plugin, when used against a libgdal not aware of it, will be systematically loaded at GDAL driver initialization time, and will not benefit from *deferred plugin loading capabilities <rfc-96>*. For that, libgdal itself must be built with the CMake variable GDAL_REGISTER_DRIVER_JP2KAK_FOR_LATER_PLUGIN=ON set.

6.75.8 See Also

- Implemented as jp2kakdataset.cpp.
- If you're using a Kakadu SDK release before v7.5, configure & compile GDAL with eg. *CXXFLAGS="-DKDU_MAJOR_VERSION=7 -DKDU_MINOR_VERSION=3 -DKDU_PATCH_VERSION=2"* for Kakadu SDK version 7.3.2.
- Alternate *JP2OpenJPEG -- JPEG2000 driver based on OpenJPEG library* driver.

6.76 JP2MrSID -- JPEG2000 via MrSID SDK

Driver short name

JP2MrSID

Build dependencies

MrSID SDK

JPEG2000 file format is supported for reading with the MrSID DSDK. It is also supported for writing with the MrSID ESDK.

JPEG2000 MrSID support is only available with the version 5.x or newer DSDK and ESDK.

6.76.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.76.2 Georeferencing

Georeferencing information can come from different sources : internal (GeoJP2 or GMLJP2 boxes), *worldfile* .j2w/.wld sidecar files, or PAM (Persistent Auxiliary metadata) .aux.xml sidecar files. By default, information is fetched in following order (first listed is the highest priority): PAM, GeoJP2, GMLJP2, WORLDFILE.

The allowed sources and their priority order can be changed with the *GDAL_GEOREF_SOURCES* configuration option (or *GEOREF_SOURCES* open option) whose value is a comma-separated list of the following keywords : PAM, GEOJP2, GMLJP2, INTERNAL (shortcut for GEOJP2,GMLJP2), WORLDFILE, NONE. Earlier mentioned sources take priority over later ones. A non mentioned source will be ignored.

For example setting it to "WORLDFILE,PAM,INTERNAL" will make a geotransformation matrix from a potential worldfile priority over PAM or internal JP2 boxes. Setting it to "PAM,WORLDFILE,GEOJP2" will use the mentioned sources and ignore GMLJP2 boxes.

6.76.3 Open Options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open option is available:

- **GEOREF_SOURCES=value:** Define which georeferencing sources are allowed and their priority order. See *Georeferencing* paragraph.

6.76.4 Creation Options

If you have the MrSID ESDK (5.x or newer), it can be used to write JPEG2000 files. Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following creation options are supported.

- **WORLDFILE=YES:** to write an ESRI world file (with the extension .j2w). See *World Files* section for details.
- **COMPRESSION=value:** Indicates the desired compression ratio. Zero indicates lossless compression. Twenty would indicate a 20:1 compression ratio (the image would be compressed to 1/20 its original size).
- **XMLPROFILE=<filename>:** Indicates a path to an Extensis-specific XML profile that can be used to set JPEG2000 encoding parameters. They can be created using the MrSID ESDK, or with GeoExpress, or by hand using the following example as a template:

```
<?xml version="1.0"?>
<Jp2Profile version="1.0">
  <Header>
    <name>Default</name>
    <description>Extensis preferred settings (20051216)</description>
  </Header>
  <Codestream>
    <layers>
```

(continues on next page)

(continued from previous page)

```
      8
    </layers>
    <levels>
      99
    </levels>
    <tileSize>
      0 0
    </tileSize>
    <progressionOrder>
      RPCL
    </progressionOrder>
    <codeblockSize>
      64 64
    </codeblockSize>
    <pltMarkers>
      true
    </pltMarkers>
    <wavelet97>
      false
    </wavelet97>
    <precinctSize>
      256 256
    </precinctSize>
  </Codestream>
</Jp2Profile>
```

6.76.5 See Also

- Implemented as mrsiddataset.cpp.
- Extensis web site

6.77 JP2OpenJPEG -- JPEG2000 driver based on OpenJPEG library

Driver short name

JP2OpenJPEG

Build dependencies

openjpeg >= 2.1

This driver is an implementation of a JPEG2000 reader/writer based on OpenJPEG library **v2**.

The driver uses the VSI Virtual File API, so it can read JPEG2000 compressed NITF files.

XMP metadata can be extracted from JPEG2000 files, and will be stored as XML raw content in the xml:XMP metadata domain.

The driver supports writing georeferencing information as GeoJP2 and GMLJP2 boxes.

The driver supports creating files with transparency, arbitrary band count, and adding/reading metadata. Update of georeferencing or metadata of existing file is also supported. Optional intellectual property metadata can be read/written in the xml:IPR box.

6.77.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.77.2 Georeferencing

Georeferencing information can come from different sources : internal (GeoJP2 or GMLJP2 boxes), worldfile .j2w/.wld sidecar files, or PAM (Persistent Auxiliary metadata) .aux.xml sidecar files. By default, information is fetched in following order (first listed is the highest priority): PAM, GeoJP2, GMLJP2, WORLDFILE.

The allowed sources and their priority order can be changed with the *GDAL_GEOREF_SOURCES* configuration option (or *GEOREF_SOURCES* open option) whose value is a comma-separated list of the following keywords : PAM, GEOJP2, GMLJP2, INTERNAL (shortcut for GEOJP2,GMLJP2), WORLDFILE, NONE. Earlier mentioned sources take priority over later ones. A non mentioned source will be ignored.

For example setting it to "WORLDFILE,PAM,INTERNAL" will make a geotransformation matrix from a potential worldfile priority over PAM or internal JP2 boxes. Setting it to "PAM,WORLDFILE,GEOJP2" will use the mentioned sources and ignore GMLJP2 boxes.

6.77.3 Thread support

By default, if the JPEG2000 file has internal tiling, GDAL will try to decode several tiles in multiple threads if the RasterIO() request it receives intersect several tiles. This behavior can be controlled with the *GDAL_NUM_THREADS* configuration option that defaults to ALL_CPUS in that context. In case RAM is limited, it can be needed to set this configuration option to 1 to disable multi-threading

Starting with OpenJPEG 2.2.0, multi-threaded decoding can also be enabled at the code-block level. This must be enabled with the OPJ_NUM_THREADS environment variable (note: this is a system environment variable, not a GDAL configuration option), which can be set to a numeric value or ALL_CPUS. Its default value is 1. Starting with GDAL 2.3, this multi-threading at code-block level is automatically enabled by GDAL. Starting with GDAL 3.0.1 and OpenJPEG 2.3.2, multi-threaded encoding is automatically enabled and can be controlled with the OPJ_NUM_THREADS environment variable or the *GDAL_NUM_THREADS* configuration option.

Both multi-threading mechanism can be combined together.

6.77.4 Open Options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are available:

- **STRICT=[YES/NO]:** (GDAL >= 3.5) Defaults to `YES`. (OpenJPEG >= 2.5) Whether strict/pedantic decoding mode should be enabled. This can be set to NO to allow decoding (some) broken files, typically truncated single-tiled files.
- **1BIT_ALPHA_PROMOTION=[YES/NO]:** Defaults to `YES`. Whether a 1-bit alpha channel should be promoted to 8-bit.
- **GEOREF_SOURCES=value:** Define which georeferencing sources are allowed and their priority order. See *Georeferencing* paragraph.
- **USE_TILE_AS_BLOCK=[YES/NO]:** Defaults to `NO`. Whether to always use the JPEG-2000 block size as the GDAL block size. Setting this option can be useful when doing whole image decompression and the image is single-tiled. Note however that the tile size must not exceed 2 GB since that's the limit supported by GDAL.

6.77.5 Creation Options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following creation options are available:

- **CODEC=[JP2/J2K]:** JP2 will add JP2 boxes around the codestream data. The value is determined automatically from the file extension. If it is neither JP2 nor J2K, J2K codec is used.
- **GMLJP2=[YES/NO]:** Defaults to `YES`. Indicates whether a GML box conforming to the OGC GML in JPEG2000 specification should be included in the file. Unless GMLJP2V2_DEF is used, the version of the GMLJP2 box will be version 1.
- **GMLJP2V2_DEF=[YES/<filename>/<json>]:** Indicates whether a GML box conforming to the OGC GML in JPEG2000, version 2.0.1 specification should be included in the file. *filename* must point to a file with a JSON content that defines how the GMLJP2 v2 box should be built. See below section for the syntax of the JSON configuration file. It is also possible to directly pass the JSON content inlined as a string. If filename is just set to YES, a minimal instance will be built.
- **GeoJP2=[YES/NO]:** Defaults to `YES`. Indicates whether a UUID/GeoTIFF box conforming to the GeoJP2 (GeoTIFF in JPEG2000) specification should be included in the file.
- **QUALITY=<float_value,float_value,...>:** Percentage between 0 and 100. A value of 50 means the file will be half-size in comparison to uncompressed data, 33 means 1/3, etc.. Defaults to 25 (unless the dataset is made of a single band with color table, in which case the default quality is 100). It is possible to specify several quality values (comma separated) to ask for several quality layers. Quality values should be increasing.
- **REVERSIBLE=[YES/NO]:** YES means use of reversible 5x3 integer-only filter, NO use of the irreversible DWT 9-7. Defaults to NO (unless the dataset is made of a single band with color table, in which case reversible filter is used).
- **RESOLUTIONS=<integer>:** Number of resolution levels. Default value is selected such the smallest overview of a tile is no bigger than 128x128.
- **BLOCKXSIZE=<integer>:** Defaults to `1024`. Tile width.
- **BLOCKYSIZE=<integer>:** Defaults to `1024`. Tile height.
- **PROGRESSION=[LRCP/RLCP/RPCL/PCRL/CPRL]:** Defaults to `LRCP`. Progression order.
- **SOP=[YES/NO]:** Defaults to `NO`. YES means generate SOP (Start Of Packet) marker segments.
- **EPH=[YES/NO]:** Defaults to `NO`. YES means generate EPH (End of Packet Header) marker segments.

- **YCBCR420=[YES/NO]:** Defaults to NO. YES if RGB must be resampled to YCbCr 4:2:0.
- **YCC=[YES/NO]:** Defaults to YES. YES if RGB must be transformed to YCC color space ("MCT transform", i.e. internal transform, without visual degradation).
- **NBITS=<integer>:** Bits (precision) for sub-byte files (1-7), sub-uint16 (9-15), sub-uint32 (17-31).
- **1BIT_ALPHA=[YES/NO]:** Whether to encode the alpha channel as a 1-bit channel (when there's an alpha channel). Defaults to NO, unless *INSPIRE_TG=YES*. Enabling this option might cause compatibility problems with some readers. At the time of writing, those based on the MrSID JPEG2000 SDK are unable to open such files. And regarding the ECW JPEG2000 SDK, decoding of 1-bit alpha channel with lossy/irreversible compression gives visual artifacts (OK with lossless encoding).
- **ALPHA=[YES/NO]:** Defaults to NO. Whether to force encoding last channel as alpha channel. Only useful if the color interpretation of that channel is not already Alpha.
- **PROFILE=[AUTO/UNRESTRICTED/PROFILE_1]:** Defaults to AUTO. Determine which codestream profile to use. UNRESTRICTED corresponds to the "Unrestricted JPEG 2000 Part 1 codestream" (RSIZ=0). PROFILE_1 corresponds to the "JPEG 2000 Part 1 Profile 1 codestream" (RSIZ=2), which add constraints on tile dimensions and number of resolutions. In AUTO mode, the driver will determine if the *BLOCKXSIZE*, *BLOCKYSIZE*, *RESOLUTIONS*, *CODEBLOCK_WIDTH* and *CODEBLOCK_HEIGHT* values are compatible with PROFILE_1 and advertise it in the relevant case. Note that the default values of those options are compatible with PROFILE_1. Otherwise UNRESTRICTED is advertised.
- **INSPIRE_TG=[YES/NO]:** Defaults to NO. Whether to use JPEG2000 features that comply with Inspire Orthoimagery Technical Guidelines. When set to YES, implies *PROFILE=PROFILE_1*, *1BIT_ALPHA=YES*, *GEOBOXES_AFTER_JP2C=YES*. The *CODEC*, *BLOCKXSIZE*, *BLOCKYSIZE*, *RESOLUTIONS*, *NBITS*, *PROFILE*, *CODEBLOCK_WIDTH* and *CODEBLOCK_HEIGHT* options will be checked against the requirements and recommendations of the Technical Guidelines.
- **JPX=[YES/NO]:** Defaults to YES. Whether to advertise JPX features, and add a Reader requirement box, when a GMLJP2 box is written. This option should not be used unless compatibility problems with a reader occur.
- **GEOBOXES_AFTER_JP2C=[YES/NO]:** Whether to place GeoJP2/GMLJP2 boxes after the code-stream. Defaults to NO, unless *INSPIRE_TG=YES*. This option should not be used unless compatibility problems with a reader occur.
- **PRECINCTS=[<{prec_w,prec_h},{prec_w,prec_h}/...>]:** A list of {precincts width,precincts height} tuples to specify precincts size. Each value should be a multiple of 2. The maximum number of tuples used will be the number of resolutions. The first tuple corresponds to the higher resolution level, and the following ones to the lower resolution levels. If less tuples are specified, the last one is used by dividing its values by 2 for each extra lower resolution level. The default value used is {512,512},{256,512},{128,512},{64,512},{32,512},{16,512},{8,512},{4,512},{2,512}. An empty string may be used to disable precincts (i.e. the default {32767,32767},{32767,32767}, ... will then be used).
- **TILEPARTS=[DISABLED/RESOLUTIONS/LAYERS/COMPONENTS]:** Defaults to DISABLED. Whether to generate tile-parts and according to which criterion.
- **CODEBLOCK_WIDTH=<integer>:** Defaults to 64. Codeblock width: power of two value between 4 and 1024. Note that CODEBLOCK_WIDTH * CODEBLOCK_HEIGHT must not be greater than 4096. For PROFILE_1 compatibility, CODEBLOCK_WIDTH must not be greater than 64.
- **CODEBLOCK_HEIGHT=<integer>:** Defaults to 64. Codeblock height: power of two value between 4 and 1024. Note that CODEBLOCK_WIDTH * CODEBLOCK_HEIGHT must not be greater than 4096. For PROFILE_1 compatibility, CODEBLOCK_HEIGHT must not be greater than 64.
- **CODEBLOCK_STYLE=value:** (OpenJPEG >= 2.3.0) Style of the code-block coding passes. The following 6 independent settings can be combined together (values should be comma separated):
  - *BYPASS* (1): enable selective arithmetic coding bypass (can substantially improve coding/decoding speed, at the expense of larger file size)

- *RESET* (2): reset context probabilities on coding pass boundaries
- *TERMALL* (4): enable termination on each coding pass
- *VSC* (8): enable vertically causal context
- *PREDICTABLE* (16): enable predictable termination (helps for error detection)
- *SEGSYM* (32): enable segmentation symbols (helps for error detection)

Instead of specifying them by text, it is also possible to give the corresponding numeric value of the global codeblock style, by adding the selected options (for example "BYPASS,TERMALL" is equivalent to "5"=1+4)

By default, none of them are enabled. Enabling them will generally increase codestream size, but improve either coding/decoding speed or resilience/error detection.

- **PLT=[YES/NO]:** (GDAL >= 3.1.1) Defaults to `NO`. (OpenJPEG >= 2.4.0) Whether to write a PLT (Packet Length) marker segment in tile-part headers.
- **TLM=[YES/NO]:** (GDAL >= 3.4.0) Defaults to `NO`. OpenJPEG >= 2.5.0) Whether to write a TLM (Tile-part Length) marker segment in main header.
- **WRITE_METADATA=[YES/NO]:** Defaults to `NO`. Whether metadata should be written, in a dedicated JP2 'xml ' box. The content of the 'xml ' box will be like:

```
<GDALMultiDomainMetadata>
  <Metadata>
    <MDI key="foo">bar</MDI>
  </Metadata>
  <Metadata domain='aux_domain'>
    <MDI key="foo">bar</MDI>
  </Metadata>
  <Metadata domain='a_xml_domain' format='xml'>
    <arbitrary_xml_content>
    </arbitrary_xml_content>
  </Metadata>
</GDALMultiDomainMetadata>
```

If there are metadata domain whose name starts with "xml:BOX_", they will be written each as separate JP2 'xml ' box.

If there is a metadata domain whose name is "xml:XMP", its content will be written as a JP2 'uuid' XMP box.

If there is a metadata domain whose name is "xml:IPR", its content will be written as a JP2 'jp2i' box.

- **MAIN_MD_DOMAIN_ONLY=[YES/NO]:** Defaults to `NO`. (Only if *`WRITE_METADATA=YES`*) Whether only metadata from the main domain should be written.
- **USE_SRC_CODESTREAM=[YES/NO]:** (EXPERIMENTAL!) When source dataset is JPEG2000, whether to reuse the codestream of the source dataset unmodified. Defaults to NO. Note that enabling that feature might result in inconsistent content of the JP2 boxes w.r.t. to the content of the source codestream. Most other creation options will be ignored in that mode. Can be useful in some use cases when adding/correcting georeferencing, metadata, ... INSPIRE_TG and PROFILE options will be ignored, and the profile of the codestream will be overridden with the one specified/implied by the options (which may be inconsistent with the characteristics of the codestream).

6.77.5.1 Lossless compression

Lossless compression can be achieved if ALL the following creation options are defined :

- *QUALITY=100*
- *REVERSIBLE=YES*
- *YCBCR420=NO* (which is the default)

6.77.5.2 GMLJP2v2 definition file

A GMLJP2v2 box typically contains a GMLJP2RectifiedGridCoverage with the SRS information and geotransformation matrix. It is also possible to add metadata, vector features (GML feature collections), annotations (KML), styles (typically SLD, or other XML format) or any XML content as an extension. The value of the GMLJP2V2_DEF creation option should be a file that conforms with the below syntax (elements starting with "#" are documentation, and can be omitted):

```
{
    "#doc" : "Unless otherwise specified, all elements are optional",

    "#root_instance_doc": "Describe content of the GMLJP2CoverageCollection",
    "root_instance": {
        "#gml_id_doc": "Specify GMLJP2CoverageCollection gml:id. Default is ID_GMLJP2_0",
        "gml_id": "some_gml_id",

        "#grid_coverage_file_doc": [
            "External XML file, whose root might be a GMLJP2GridCoverage, ",
            "GMLJP2RectifiedGridCoverage or a GMLJP2ReferenceableGridCoverage.",
            "If not specified, GDAL will auto-generate a GMLJP2RectifiedGridCoverage" ],
        "grid_coverage_file": "gmljp2gridcoverage.xml",

        "#grid_coverage_range_type_field_predefined_name_doc": [
            "One of Color, Elevation_meter or Panchromatic ",
            "to fill gmlcov:rangeType/swe:DataRecord/swe:field",
            "Only used if grid_coverage_file is not defined.",
            "Exclusive with grid_coverage_range_type_file" ],
        "grid_coverage_range_type_field_predefined_name": "Color",

        "#grid_coverage_range_type_file_doc": [
            "File that is XML content to put under gml:RectifiedGrid/gmlcov:rangeType",
            "Only used if grid_coverage_file is not defined.",
            "Exclusive with grid_coverage_range_type_field_predefined_name" ],
        "grid_coverage_range_type_file": "grid_coverage_range_type.xml",

        "#crs_url_doc": [
            "true for http://www.opengis.net/def/crs/EPSG/0/XXXX CRS URL.",
            "If false, use CRS URN. Default value is true",
            "Only taken into account for a auto-generated GMLJP2RectifiedGridCoverage"],
        "crs_url": true,

        "#metadata_doc": [ "An array of metadata items. Can be either strings, with ",
                           "a filename or directly inline XML content, or either ",
                           "a more complete description." ],
        "metadata": [
```

(continues on next page)

(continued from previous page)

```
            "dcmetadata.xml",

            {
                "#file_doc": "Can use relative or absolute paths. Exclusive of content,␣
↪gdal_metadata and generated_metadata.",
                "file": "dcmetadata.xml",

                "#gdal_metadata_doc": "Whether to serialize GDAL metadata as␣
↪GDALMultiDomainMetadata",
                "gdal_metadata": false,

                "#dynamic_metadata_doc":
                    [ "The metadata file will be generated from a template and a source␣
↪file.",
                      "The template is a valid GMLJP2 metadata XML tree with␣
↪placeholders like",
                      "{{{XPATH(some_xpath_expression)}}}",
                      "that are evaluated from the source XML file. Typical use case",
                      "is to generate a gmljp2:eopMetadata from the XML metadata",
                      "provided by the image provider in their own particular format." ],
                "dynamic_metadata" :
                {
                    "template": "my_template.xml",
                    "source": "my_source.xml"
                },

                "#content": "Exclusive of file. Inline XML metadata content",
                "content": "<gmljp2:metadata>Some simple textual metadata</
↪gmljp2:metadata>",

                "#parent_node": ["Where to put the metadata.",
                                 "Under CoverageCollection (default) or GridCoverage" ],
                "parent_node": "CoverageCollection"
            }
        ],

        "#annotations_doc": [ "An array of filenames, either directly KML files",
                              "or other vector files recognized by GDAL that ",
                              "will be translated on-the-fly as KML" ],
        "annotations": [
            "my.kml"
        ],

        "#gml_filelist_doc" :[
            "An array of GML files or vector files that will be on-the-fly converted",
            "to GML 3.2. Can be either GML filenames (or other OGR datasource names), ",
            "or a more complete description" ],
        "gml_filelist": [

            "my.gml",

            "my.shp",
```

(continues on next page)

(continued from previous page)

```
            {
                "#file_doc": "OGR datasource. Can use relative or absolute paths.␣
↪Exclusive of remote_resource",
                "file": "converted/test_0.gml",

                "#remote_resource_doc": "URL of a feature collection that must be␣
↪referenced through a xlink:href",
                "remote_resource": "https://github.com/OSGeo/gdal/blob/master/autotest/
↪ogr/data/expected_gml_gml32.gml",

                "#namespace_doc": ["The namespace in schemaLocation for which to␣
↪substitute",
                                   "its original schemaLocation with the one provided␣
↪below.",
                                   "Ignored for a remote_resource"],
                "namespace": "http://example.com",

                "#schema_location_doc": ["Value of the substituted schemaLocation. ",
                                         "Typically a schema box label (link)",
                                         "Ignored for a remote_resource"],
                "schema_location": "gmljp2://xml/schema_0.xsd",

                "#inline_doc": [
                    "Whether to inline the content, or put it in a separate xml box.␣
↪Default is true",
                    "Ignored for a remote_resource." ],
                "inline": true,

                "#parent_node": ["Where to put the FeatureCollection.",
                                 "Under CoverageCollection (default) or GridCoverage" ],
                "parent_node": "CoverageCollection"
            }
        ],


        "#styles_doc": [ "An array of styles. For example SLD files" ],
        "styles" : [
            {
                "#file_doc": "Can use relative or absolute paths.",
                "file": "my.sld",

                "#parent_node": ["Where to put the FeatureCollection.",
                                 "Under CoverageCollection (default) or GridCoverage" ],
                "parent_node": "CoverageCollection"
            }
        ],

        "#extensions_doc": [ "An array of extensions." ],
        "extensions" : [
            {
                "#file_doc": "Can use relative or absolute paths.",
```

(continues on next page)

(continued from previous page)

```
                "file": "my.xml",

                "#parent_node": ["Where to put the FeatureCollection.",
                                 "Under CoverageCollection (default) or GridCoverage" ],
                "parent_node": "CoverageCollection"
            }
        ]
    },

    "#boxes_doc": "An array to describe the content of XML asoc boxes",
    "boxes": [
        {
            "#file_doc": "can use relative or absolute paths. Required",
            "file": "converted/test_0.xsd",

            "#label_doc": ["the label of the XML box. If not specified, will be the ",
                           "filename without the directory part." ],
            "label": "schema_0.xsd"
        }
    ]
}
```

Metadata can be dynamically generated from a template file (in that context, with a XML structure) and a XML source file. The template file is processed by searching for patterns like {{{XPATH(xpath_expr)}}} and replacing them by their evaluation against the content of the source file. xpath_expr must be a XPath 1.0 compatible expression, with the addition of the following functions :

- **if(cond_expr,expr_if_true,expr_if_false)**: if cond_expr evaluates to true, returns expr_if_true. Otherwise returns expr_if_false
- **uuid()**: evaluates to a random UUID

A template file to process XML metadata of Pleiades imagery can be found here, and a template file to process XML metadata of GeoEye/WorldView imagery can be found here.

6.77.6 Vector information

A JPEG2000 file containing a GMLJP2 v2 box with GML feature collections and/or KML annotations embedded can be opened as a vector file with the OGR API. For example:

```
ogrinfo -ro my.jp2

INFO: Open of my.jp2'
      using driver `JP2OpenJPEG' successful.
1: FC_GridCoverage_1_rivers (LineString)
2: FC_GridCoverage_1_borders (LineString)
3: Annotation_1_poly
```

Feature collections can be linked from the GMLJP2 v2 box to a remote location. By default, the link is not followed. It will be followed if the open option OPEN_REMOTE_GML is set to YES.

6.77.7 Standalone plugin compilation

Added in version 3.10.

While this driver may be built as part of a whole GDAL build, either in libgdal itself, or as a plugin, it is also possible to only build this driver as a plugin, against an already built libgdal.

The version of the GDAL sources used to build the driver must match the version of the libgdal it is built against.

For example, from a "build_openjpeg" directory under the root of the GDAL source tree:

```
cmake -S ../frmts/openjpeg -DCMAKE_PREFIX_PATH=/path/to/GDAL_installation_prefix -
↪DOPENJPEG_ROOT=/path/to/openjpeg_install_prefix
cmake --build .
```

Note that such a plugin, when used against a libgdal not aware of it, will be systematically loaded at GDAL driver initialization time, and will not benefit from *deferred plugin loading capabilities <rfc-96>*. For that, libgdal itself must be built with the CMake variable GDAL_REGISTER_DRIVER_JP2OPENJPEG_FOR_LATER_PLUGIN=ON set.

6.77.8 See Also

- Implemented as openjpegdataset.cpp.
- Official JPEG-2000 page
- The OpenJPEG library home page
- OGC GML in JPEG2000, version 2.0.1
- Inspire Data Specification on Orthoimagery - Technical Guidelines

Other JPEG2000 GDAL drivers :

- *JP2ECW: based on Erdas ECW library (proprietary)*
- *JP2MRSID: based on LizardTech MrSID library (proprietary)*
- *JP2KAK: based on Kakadu library (proprietary)*

6.78 JPEG -- JPEG JFIF File Format

Driver short name

JPEG

Build dependencies

(internal libjpeg provided)

The JPEG JFIF format is supported for reading, and batch writing, but not update in place. JPEG files are represented as one band (greyscale) or three band (RGB) datasets with Byte valued bands.

The driver will automatically convert images whose color space is YCbCr, CMYK or YCbCrK to RGB, unless *GDAL_JPEG_TO_RGB* is set to NO (YES is the default). When color space translation to RGB is done, the source color space is indicated in the SOURCE_COLOR_SPACE metadata of the IMAGE_STRUCTURE domain.

EXIF metadata can be read from JPEG files (but this will not result in a georeferenced image even if the EXIF_GPSLatitude and EXIF_GPSLongitude tags are set). But if an ESRI *world file* exists with the .jgw, .jpgw/.jpegw or .wld suffixes, it will be read and used to establish the geotransform for the image. If available a MapInfo .tab file will also be used for georeferencing. Overviews can be built for JPEG files as an external .ovr file.

The driver also supports the "zlib compressed mask appended to the file" approach used by a few data providers to add a bitmask to identify pixels that are not valid data. See rfc-15 for further details.

The driver can deal with bitmask where the bits are ordered with most significant bit first (whereas the usual convention is least significant bit first). The driver will try to autodetect that situation, but the heuristics may fail. In that circumstance, you can set the `JPEG_MASK_BIT_ORDER` configuration option to MSB. Bitmask can also be completely ignored by specifying `JPEG_READ_MASK=NO`.

The GDAL JPEG Driver is built using the Independent JPEG Group's jpeg library. Also note that the GeoTIFF driver supports tiled TIFF with JPEG compressed tiles. This can be used to apply JPEG compression to datasets that exceed the maximum dimensions of 65,535x65,535 pixels for a single JPEG image.

It is also possible to use the JPEG driver with the libjpeg-turbo, a version of libjpeg, API and ABI compatible with IJG libjpeg-6b, which uses MMX, SSE, and SSE2 SIMD instructions to accelerate baseline JPEG compression/decompression.

Starting with GDAL 3.4, read and write support for JPEG images with 12-bit sample is enabled by default (if JPEG support is also enabled), using GDAL internal libjpeg (based on IJG libjpeg-6b, with additional changes for 12-bit sample support). Support for JPEG with 12-bit sample is independent of whether 8-bit JPEG support is enabled through internal IJG libjpeg-6b or external libjpeg (like libjpeg-turbo)

XMP metadata can be extracted from the file, and will be stored as XML raw content in the xml:XMP metadata domain.

Embedded EXIF thumbnails (with JPEG compression) can be used as overviews, and generated by GDAL.

6.78.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.78.2 Color Profile Metadata

GDAL can deal with the following color profile metadata in the COLOR_PROFILE domain:

- SOURCE_ICC_PROFILE (Base64 encoded ICC profile embedded in file.)

Note that this metadata property can only be used on the original raw pixel data. If automatic conversion to RGB has been done, the color profile information cannot be used.

This metadata tag can be used as creation options.

6.78.3 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). The following configuration options are available :

- **GDAL_JPEG_TO_RGB=[YES/NO]:** Whether CMYK JPEG images should be exposed as RGB.
- **JPEG_READ_MASK=[YES/NO]:** Defaults to `YES`. Whether to read the bitmask identifying pixels with valid data.
- **JPEG_MASK_BIT_ORDER=[AUTO/LSB/MSB]:** Defaults to `AUTO`. Specifies if the mask is written with the least-significant bit (LSB) first, with the most-significant bit (MSB) first, or if GDAL should try and detect the bit order.
- **GDAL_ERROR_ON_LIBJPEG_WARNING=[TRUE/FALSE]:** Defaults to `FALSE`. While decoding, libjpeg has resiliency towards some errors in the JPEG datastream and will try to recover from them as much of possible. Such errors will be reported as GDAL Warnings, but can optionally be considered as true Errors by setting the *GDAL_ERROR_ON_LIBJPEG_WARNING* configuration option to TRUE.

6.78.4 Open Options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are available:

- **USE_INTERNAL_OVERVIEWS=[YES/NO]:** Defaults to `YES`. Whether to use partial DCT decompression to generate overviews.
- **APPLY_ORIENTATION=[YES/NO]:** (GDAL >= 3.7) Defaults to `NO`. Whether to use EXIF_Orientation metadata item to rotate/flip the image to apply scene orientation. Defaults to NO (that is the image will be returned in sensor orientation).

6.78.5 Creation Options

JPEG files are created using the "JPEG" driver code. Only Byte band types are supported.

Only 1 (greyscale), 3 band (input should be in RGB colorspace. the driver will convert it automatically to YCbCr colorspace for storage, and will expose it back as RGB on reading) or 4 band (input should already by in CMYK colorspace. It will be exposed as RGB on reading by default, unless the *GDAL_JPEG_TO_RGB* configuration option is set to NO) configurations.

JPEG file creation is implemented by the batch (CreateCopy) method. YCbCrK colorspace is not supported in creation. If the source dataset has a nodata mask, it will be appended as a zlib compressed mask to the JPEG file.

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following creation options are supported:

- **WORLDFILE=[YES/NO]:** Defaults to `NO`. Force the generation of an associated ESRI world file (with the extension .wld). See *World Files* section for details.
- **QUALITY=1-100:** Defaults to `75`. Set the JPEG quality. Low values result in higher compression ratios, but poorer image quality with strong blocking artifacts. Values above 95 are not meaningfully better quality but can be substantially larger.
- **LOSSLESS_COPY=[AUTO/YES/NO]:** (GDAL >= 3.7) Defaults to `AUTO`. Whether conversion should be lossless. In AUTO or YES mode, if the source dataset uses JPEG compression, or JPEGXL compression with a JPEG reconstruction box, lossless recoding from it is done. If set to NO, or in AUTO mode and the source is not compatible with lossless transcoding, the regular conversion code path is taken.

- **PROGRESSIVE=ON:** Enabled generation of progressive JPEGs. In some cases these will display a reduced resolution image in viewers such as Netscape, and Internet Explorer, before the full file has been downloaded. However, some applications cannot read progressive JPEGs at all. GDAL can read progressive JPEGs, but takes no advantage of their progressive nature.
- **INTERNAL_MASK=[YES/NO]:** By default, if needed, an internal mask in the "zlib compressed mask appended to the file" approach is written to identify pixels that are not valid data. This can be disabled by setting this option to NO.
- **ARITHMETIC=[YES/NO]:** To enable arithmetic coding. Not enabled in all libjpeg builds, because of possible legal restrictions.
- **BLOCK=1-16:** Defaults to `8`. (libjpeg >= 8c) DCT block size. All values from 1 to 16 are possible. Default is 8 (baseline format). A value other than 8 will produce files incompatible with versions prior to libjpeg 8c.
- **COLOR_TRANSFORM=[RGB/RGB1]:** (libjpeg >= 9). Set to RGB1 for lossless RGB. Note: this will produce files incompatible with versions prior to libjpeg 9.
- **SOURCE_ICC_PROFILE=value:** ICC profile encoded in Base64.
- **COMMENT=value:** String to embed in a comment JPEG marker. When reading, such strings are exposed in the COMMENT metadata item.
- **EXIF_THUMBNAIL=[YES/NO]:** Defaults to `NO`. Whether to generate an EXIF thumbnail(overview), itself JPEG compressed. If enabled, the maximum dimension of the thumbnail will be 128, if neither THUMBNAIL_WIDTH nor THUMBNAIL_HEIGHT are specified.
- **THUMBNAIL_WIDTH=value:** Width of thumbnail. Only taken into account if *`EXIF_THUMBNAIL=YES`*.
- **THUMBNAIL_HEIGHT=value:** Height of thumbnail. Only taken into account if *`EXIF_THUMBNAIL=YES`*.
- **WRITE_EXIF_METADATA=[YES/NO]:** Defaults to `YES`. Whether to write EXIF_xxxx metadata items in a EXIF segment.

6.78.6 EXIF and GPS tags

The below tables list the EXIF and GPS tags that can be written.

- The "Metadata item name" column presents the name of the metadata item to attach to the source dataset.
- The "Hex code" column is the value of the corresponding TIFF EXIF/GPS tag (for reference only)
- The "Type" column is the TIFF type associated.
  - ASCII is for text values that are NUL-terminated (for a fixed length tag, the length includes this NUL-terminating characters). e.g EXIF_Make=the_make
  - BYTE/UNDEFINED is for values that can be made of any byte value. The value of the corresponding GDAL metadata item must be a string of hexadecimal formatted values, e.g EXIF_GPSVersionID=0x02 0x00 0x00 0x00. GDAL also accepts an ASCII string: e.g. EXIF_ExifVersion=0231
  - SHORT is for unsigned integer values in the range [0,65535]. Some tags may accept multiple values, in which case they must be separated by space.
  - LONG is for unsigned integer values in the range [0,4294967295]. Some tags may accept multiple values, in which case they must be separated by space.
  - RATIONAL is for positive floating-point values. Some tags may accept multiple values, in which case they must be separated by space. e.g EXIF_GPSLatitude=49 2 3.5
  - SRATIONAL is for positive or negative floating-point values. Some tags may accept multiple values, in which case they must be separated by space.

When an item accepts a fixed number of values and that more are provided, they will be truncated with a warning. In the case they are less values provided than needed, they will be padded with appropriate spaces / zeroes

- The "Number of values" column is the number of values for the item. Might be "variable" if there is no restriction, or a fixed value. For Type=ASCII, the fixed value includes the NUL-terminating byte, so the number of actual printable characters is number of values - 1.
- The "Optionality" column indicates whether the item should be present ("Mandatory"), is "Recommended" or "Optional". GDAL does not enforce this.

Many items have more restrictions on the valid content that are not expressed in the below tables. Consult the EXIF specification for more information.

Metadata item name	Hex code	Type	Number of values	Optionality
EXIF_Document_Name	0x010D	ASCII	variable	Optional
EXIF_ImageDescription	0x010E	ASCII	variable	Recommended
EXIF_Make	0x010F	ASCII	variable	Recommended
EXIF_Model	0x0110	ASCII	variable	Recommended
EXIF_Orientation	0x0112	SHORT	1	Recommended
EXIF_XResolution	0x011A	RATIONAL	1	**Mandatory**
EXIF_YResolution	0x011B	RATIONAL	1	**Mandatory**
EXIF_ResolutionUnit	0x0128	SHORT	1	**Mandatory**
EXIF_TransferFunction	0x012D	SHORT	768	Optional
EXIF_Software	0x0131	ASCII	variable	Optional
EXIF_DateTime	0x0132	ASCII	20	Recommended
EXIF_Artist	0x013B	ASCII	variable	Optional
EXIF_WhitePoint	0x013E	RATIONAL	2	Optional
EXIF_PrimaryChromaticities	0x013F	RATIONAL	6	Optional
EXIF_YCbCrCoefficients	0x0211	RATIONAL	3	Optional
EXIF_YCbCrPositioning	0x0213	SHORT	1	**Mandatory**
EXIF_ReferenceBlackWhite	0x0214	RATIONAL	6	Optional
EXIF_Copyright	0x8298	ASCII	variable	Optional
EXIF_ExposureTime	0x829A	RATIONAL	1	Recommended
EXIF_FNumber	0x829D	RATIONAL	1	Optional
EXIF_ExposureProgram	0x8822	SHORT	1	Optional
EXIF_SpectralSensitivity	0x8824	ASCII	variable	Optional
EXIF_ISOSpeedRatings	0x8827	SHORT	variable	Optional
EXIF_OECF	0x8828	UNDEFINED	variable	Optional
EXIF_SensitivityType	0x8830	SHORT	1	Optional
EXIF_StandardOutputSensitivity	0x8831	LONG	1	Optional
EXIF_RecommendedExposureIndex	0x8832	LONG	1	Optional
EXIF_ISOSpeed	0x8833	LONG	1	Optional
EXIF_ISOSpeedLatitudeyyy	0x8834	LONG	1	Optional
EXIF_ISOSpeedLatitudezzz	0x8835	LONG	1	Optional
EXIF_ExifVersion	0x9000	UNDEFINED	4	**Mandatory**
EXIF_DateTimeOriginal	0x9003	ASCII	20	Optional
EXIF_DateTimeDigitized	0x9004	ASCII	20	Optional
EXIF_OffsetTime	0x9010	ASCII	7	Optional
EXIF_OffsetTimeOriginal	0x9011	ASCII	7	Optional
EXIF_OffsetTimeDigitized	0x9012	ASCII	7	Optional
EXIF_ComponentsConfiguration	0x9101	UNDEFINED	4	**Mandatory**
EXIF_CompressedBitsPerPixel	0x9102	RATIONAL	1	Optional
EXIF_ShutterSpeedValue	0x9201	SRATIONAL	1	Optional
EXIF_ApertureValue	0x9202	RATIONAL	1	Optional

continues on next page

Table 4 – continued from previous page

Metadata item name	Hex code	Type	Number of values	Optionality
EXIF_BrightnessValue	0x9203	SRATIONAL	1	Optional
EXIF_ExposureBiasValue	0x9204	SRATIONAL	1	Optional
EXIF_MaxApertureValue	0x9205	RATIONAL	1	Optional
EXIF_SubjectDistance	0x9206	RATIONAL	1	Optional
EXIF_MeteringMode	0x9207	SHORT	1	Optional
EXIF_LightSource	0x9208	SHORT	1	Optional
EXIF_Flash	0x9209	SHORT	1	Recommended
EXIF_FocalLength	0x920A	RATIONAL	1	Optional
EXIF_SubjectArea	0x9214	SHORT	variable	Optional
EXIF_MakerNote	0x927C	UNDEFINED	variable	Optional
EXIF_UserComment	0x9286	UNDEFINED	variable	Optional
EXIF_SubSecTime	0x9290	ASCII	variable	Optional
EXIF_SubSecTime_Original	0x9291	ASCII	variable	Optional
EXIF_SubSecTime_Digitized	0x9292	ASCII	variable	Optional
EXIF_FlashpixVersion	0xA000	UNDEFINED	4	**Mandatory**
EXIF_ColorSpace	0xA001	SHORT	1	**Mandatory**
EXIF_PixelXDimension	0xA002	LONG	1	**Mandatory**
EXIF_PixelYDimension	0xA003	LONG	1	**Mandatory**
EXIF_RelatedSoundFile	0xA004	ASCII	13	Optional
EXIF_FlashEnergy	0xA20B	RATIONAL	1	Optional
EXIF_SpatialFrequencyResponse	0xA20C	UNDEFINED	variable	Optional
EXIF_FocalPlaneXResolution	0xA20E	RATIONAL	1	Optional
EXIF_FocalPlaneYResolution	0xA20F	RATIONAL	1	Optional
EXIF_FocalPlaneResolutionUnit	0xA210	SHORT	1	Optional
EXIF_SubjectLocation	0xA214	SHORT	2	Optional
EXIF_ExposureIndex	0xA215	RATIONAL	1	Optional
EXIF_SensingMethod	0xA217	SHORT	1	Optional
EXIF_FileSource	0xA300	UNDEFINED	1	Optional
EXIF_SceneType	0xA301	UNDEFINED	1	Optional
EXIF_CFAPattern	0xA302	UNDEFINED	variable	Optional
EXIF_CustomRendered	0xA401	SHORT	1	Optional
EXIF_ExposureMode	0xA402	SHORT	1	Recommended
EXIF_WhiteBalance	0xA403	SHORT	1	Recommended
EXIF_DigitalZoomRatio	0xA404	RATIONAL	1	Optional
EXIF_FocalLengthIn35mmFilm	0xA405	SHORT	1	Optional
EXIF_SceneCaptureType	0xA406	SHORT	1	Recommended
EXIF_GainControl	0xA407	RATIONAL	1	Optional
EXIF_Contrast	0xA408	SHORT	1	Optional
EXIF_Saturation	0xA409	SHORT	1	Optional
EXIF_Sharpness	0xA40A	SHORT	1	Optional
EXIF_DeviceSettingDescription	0xA40B	UNDEFINED	variable	Optional
EXIF_SubjectDistanceRange	0xA40C	SHORT	1	Optional
EXIF_ImageUniqueID	0xA420	ASCII	33	Optional
EXIF_CameraOwnerName	0xA430	ASCII	variable	Optional
EXIF_BodySerialNumber	0xA431	ASCII	variable	Optional
EXIF_LensSpecification	0xA432	RATIONAL	4	Optional
EXIF_LensMake	0xA433	ASCII	variable	Optional
EXIF_LensModel	0xA434	ASCII	variable	Optional
EXIF_LensSerialNumber	0xA435	ASCII	variable	Optional

GPS tags:

Metadata item name	Hex code	Type	Number of values	Optionality
EXIF_GPSVersionID	0x0000	BYTE	4	Optional
EXIF_GPSLatitudeRef	0x0001	ASCII	2	Optional
EXIF_GPSLatitude	0x0002	RATIONAL	3	Optional
EXIF_GPSLongitudeRef	0x0003	ASCII	2	Optional
EXIF_GPSLongitude	0x0004	RATIONAL	3	Optional
EXIF_GPSAltitudeRef	0x0005	BYTE	1	Optional
EXIF_GPSAltitude	0x0006	RATIONAL	1	Optional
EXIF_GPSTimeStamp	0x0007	RATIONAL	3	Optional
EXIF_GPSSatellites	0x0008	ASCII	variable	Optional
EXIF_GPSStatus	0x0009	ASCII	2	Optional
EXIF_GPSMeasureMode	0x000A	ASCII	2	Optional
EXIF_GPSDOP	0x000B	RATIONAL	1	Optional
EXIF_GPSSpeedRef	0x000C	ASCII	2	Optional
EXIF_GPSSpeed	0x000D	RATIONAL	1	Optional
EXIF_GPSTrackRef	0x000E	ASCII	2	Optional
EXIF_GPSTrack	0x000F	RATIONAL	1	Optional
EXIF_GPSImgDirectionRef	0x0010	ASCII	2	Optional
EXIF_GPSImgDirection	0x0011	RATIONAL	1	Optional
EXIF_GPSMapDatum	0x0012	ASCII	variable	Optional
EXIF_GPSDestLatitudeRef	0x0013	ASCII	2	Optional
EXIF_GPSDestLatitude	0x0014	RATIONAL	3	Optional
EXIF_GPSDestLongitudeRef	0x0015	ASCII	2	Optional
EXIF_GPSDestLongitude	0x0016	RATIONAL	3	Optional
EXIF_GPSDestBearingRef	0x0017	ASCII	2	Optional
EXIF_GPSDestBearing	0x0018	RATIONAL	1	Optional
EXIF_GPSDestDistanceRef	0x0019	ASCII	2	Optional
EXIF_GPSDestDistance	0x001A	RATIONAL	1	Optional
EXIF_GPSProcessingMethod	0x001B	UNDEFINED	variable	Optional
EXIF_GPSAreaInformation	0x001C	UNDEFINED	variable	Optional
EXIF_GPSDateStamp	0x001D	ASCII	11	Optional
EXIF_GPSDifferential	0x001E	SHORT	1	Optional
EXIF_GPSHPositioningError	0x001F	RATIONAL	1	Optional

6.78.7 FLIR metadata

Added in version 3.3.

Metadata encoded according to the FLIR convention (infrared images) is available in the `FLIR` metadata domain.

Metadata from the following sections is supported:

- Header
- RawData
- CameraInfo
- PaletteInfo
- GPSInfo

Consult https://exiftool.org/TagNames/FLIR.html for details.

The thermal image data, stored either as raw data or in PNG, is exposed as a GDAL subdataset whose name is `JPEG:"filename.jpg":FLIR_RAW_THERMAL_IMAGE`

6.78.8 See Also

- Independent JPEG Group
- libjpeg-turbo
- *GTiff -- GeoTIFF File Format*
- EXIF v2.31 specification

6.79 JPEGXL -- JPEG-XL File Format

Added in version 3.6.

Driver short name

JPEGXL

Build dependencies

libjxl

The JPEG-XL format is supported for reading, and batch writing (CreateCopy()), but not update in place.

The driver supports reading and writing: - georeferencing: encoded as a GeoJP2 UUID box within a JUMBF box. - XMP in the xml:XMP metadata domain - EXIF in the EXIF metadata domain - color profile in the COLOR_PROFILE metadata domain.

Reading or writing involves ingesting the whole uncompressed image in memory. Compression is in particular very memory hungry with current libjxl implementation. For large images (let's say width or height larger than 10,000 pixels), using JPEGXL compression as a *GTiff -- GeoTIFF File Format* codec is thus recommended.

The number of worker threads for multi-threaded compression and decompression can be set with the `GDAL_NUM_THREADS` configuration option to an integer value or `ALL_CPUS` (the later is the default).

Note

Support for reading and writing XMP and EXIF, and writing georeferencing, requires a libjxl version build from its main branch, post 0.6.1 release.

6.79.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.79.2 Color Profile Metadata

GDAL can deal with the following color profile metadata in the COLOR_PROFILE domain:

- SOURCE_ICC_PROFILE (Base64 encoded ICC profile embedded in file.)

6.79.3 Open Options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are available:

- **APPLY_ORIENTATION=[YES/NO]:** (GDAL >= 3.7) Defaults to `NO`. Whether to use EXIF_Orientation metadata item to rotate/flip the image to apply scene orientation. Defaults to NO (that is the image will be returned in sensor orientation).

6.79.4 Creation Options

With libjxl 0.6.1, only 1 (greyscale), 2 band (greyscale + alpha), 3 band (RGB) or 4 band (RGBA) source images are supported. With later libjxl versions, any number of bands can be written. Supported data types are Byte, UInt16 and Float32.

When copying from a (regular) JPEG file, and not specifying lossy compression options, its content is re-encoded in a lossless way, and with reconstruction data that enables to recreate a JPEG file from the JPEGXL codestream.

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following creation options are available:

- **LOSSLESS=[YES/NO]:** Whether JPEGXL compression should be lossless. Defaults to YES (unless DISTANCE or QUALITY are specified)
- **LOSSLESS_COPY=[AUTO/YES/NO]:** (GDAL >= 3.7) Defaults to `AUTO`. Whether conversion should be lossless. In AUTO or YES mode, if LOSSLESS=YES and the source dataset uses JPEG compression, lossless recoding of it to JPEGXL is done, and a JPEG reconstruction box is added so that reverse conversion to JPEG is possible. If set to NO, or in AUTO mode if the source dataset does not use JPEG compression, the regular conversion code path is taken, resulting in a lossless or lossy copy depending on the LOSSLESS setting. AUTO mode defaults to NO, if EFFORT, DISTANCE, ALPHA_DISTANCE or QUALITY options are used.
- **EFFORT=1-9:** Defaults to `5`. Level of effort. The higher, the smaller file and slower compression time.
- **DISTANCE=0.01-25:** Defaults to `1.0`. Distance level for lossy JPEG-XL compression. It is specified in multiples of a just-noticeable difference. (cf butteraugli for the definition of the distance) That is, 0 is mathematically lossless, 1 should be visually lossless, and higher distances yield denser and denser files with lower and lower fidelity. The recommended range is [0.5,3].
- **ALPHA_DISTANCE=[-1/0/0.01-25]:** (GDAL >= 3.7) Defaults to `-1.0`. (libjxl > 0.8.1) Distance level for alpha channel for lossy JPEG-XL compression. It is specified in multiples of a just-noticeable difference. (cf butteraugli for the definition of the distance) That is, 0 is mathematically lossless, 1 should be visually lossless, and higher distances yield denser and denser files with lower and lower fidelity. For lossy compression, the recommended range is [0.5,3]. The default value is the special value -1.0, which means to use the same distance value as non-alpha channel (ie DISTANCE).
- **QUALITY=[-inf,100]:** Defaults to `90.0`. Alternative setting to *`DISTANCE`* to specify lossy compression, roughly matching libjpeg quality setting in the [0,100] range.

- **NBITS=<integer>:** Create a file with less than 8 bits per sample by passing a value from 1 to 7 for a Byte type, or a value from 9 to 15 for a UInt16 type.
- **NUM_THREADS=[<number_of_threads>/ALL_CPUS]:** Defaults to `ALL_CPUS`. Set the number of worker threads for multi-threaded compression. If not set, can also be controlled with the *GDAL_NUM_THREADS* configuration option.
- **SOURCE_ICC_PROFILE=value:** ICC profile encoded in Base64. Can also be set to empty string to avoid the ICC profile from the source dataset to be used.
- **WRITE_EXIF_METADATA=[YES/NO]:** Defaults to `YES`. (libjxl > 0.6.1) Whether to write EXIF metadata from the EXIF metadata domain of the source dataset in a Exif box.
- **WRITE_XMP=[YES/NO]:** Defaults to `YES`. (libjxl > 0.6.1) Whether to write XMP metadata from the xml:XMP metadata domain of the source dataset in a xml box.
- **WRITE_GEOJP2=[YES/NO]:** Defaults to `YES`. (libjxl > 0.6.1) Whether to write georeferencing in a JUMBF UUID box using GeoJP2 encoding.
- **COMPRESS_BOXES=[YES/NO]:** Defaults to `NO`. (libjxl > 0.6.1) Whether to to decompress Exif/XMP/GeoJP2 boxes using Brotli compression.

6.79.5 See Also

- JPEG-XL home page
- libjxl

6.80 JPIPKAK - JPIP Streaming

Driver short name

JPIPKAK

Build dependencies

Kakadu library

JPEG 2000 Interactive Protocol (JPIP) flexibility with respect to random access, code stream reordering and incremental decoding is highly exploitable in a networked environment allowing access to remote large files using limited bandwidth connections or high contention networks.

6.80.1 Driver capabilities

Supports Georeferencing

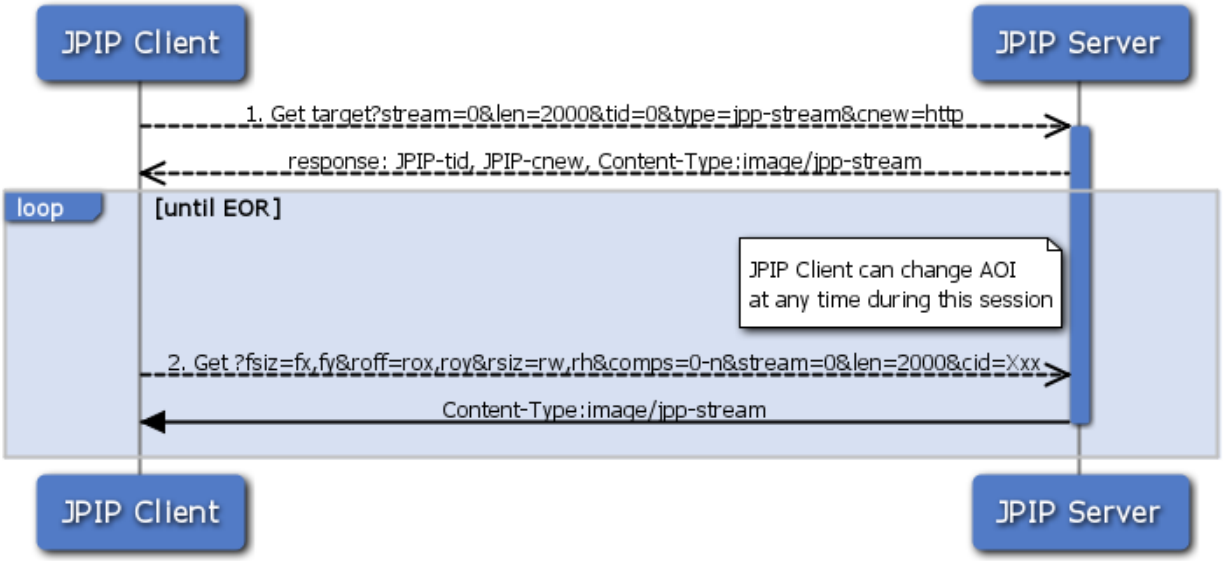
This driver supports georeferencing

6.80.2 JPIPKAK - JPIP Overview

A brief overview of the JPIP event sequence is presented in this section, more information can be found at JPEG 2000 Interactive Protocol (Part 9 – JPIP) and the specification can (and should) be purchased from ISO.

An earlier version of JPEG 2000 Part 9 is available here http://www.jpeg.org/public/fcd15444-9v2.pdf, noting the ISO copyright, diagrams are not replicated in this documentation.

The JPIP protocol has been abstracted in this format driver, requests are made at the 1:1 resolution level.



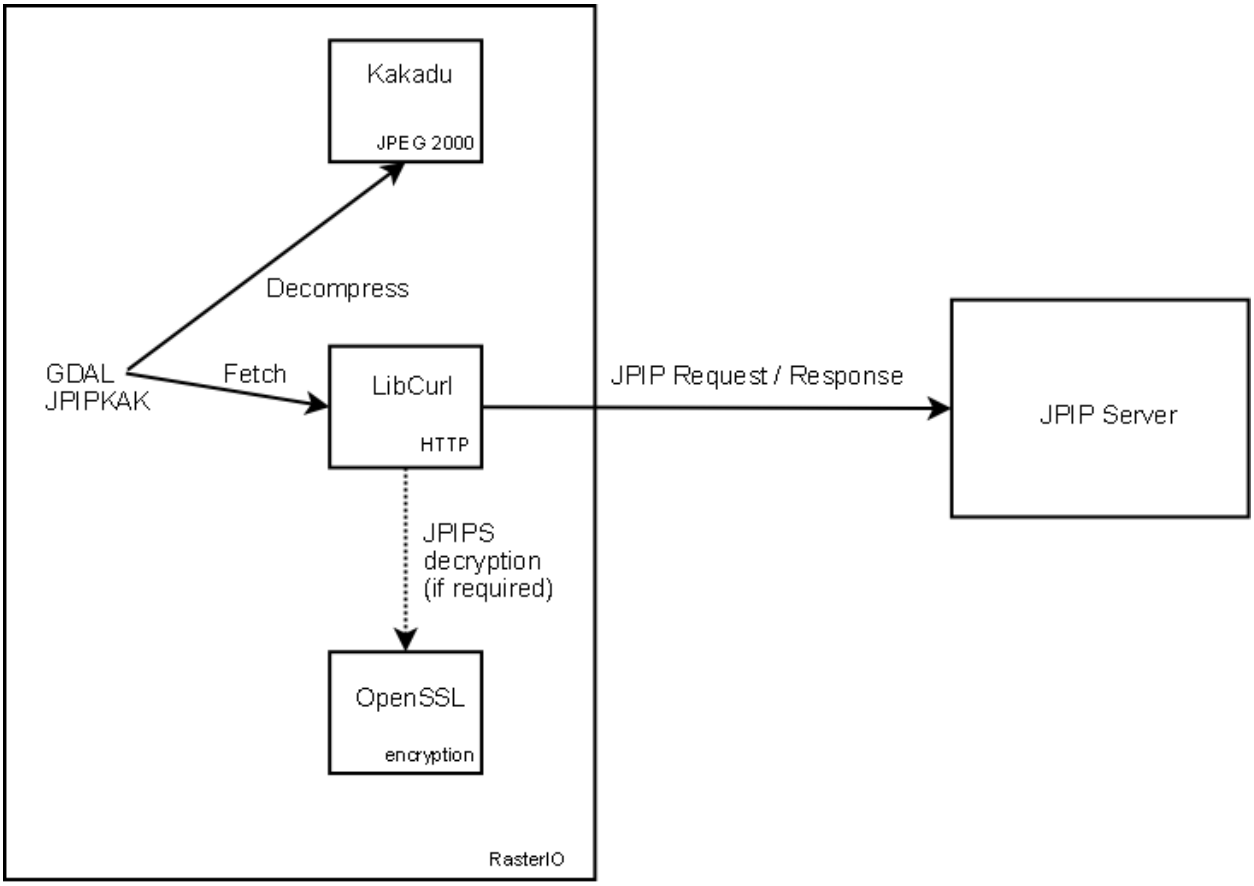
1. Initial JPIP request for a target image, a target id, a session over http, data to be returned as a jpp-stream are requested and a maximum length is put on the response. In this case no initial window is requested, though it can be. Server responds with a target identifier that can be used to identify the image on the server and a JPIP-cnew response header which includes the path to the JPIP server which will handle all future requests and a cid session identifier. A session is required so that that the server can model the state of the client connection, only sending the data that is required.
2. Client requests particular view windows on the target image with a maximum response length and includes the session identifier established in the previous communication. 'fsiz' is used to identify the resolution associated with the requested view-window. The values 'fx' and 'fy' specify the dimensions of the desired image resolution. 'roff' is used to identify the upper left hand corner off the spatial region associated with the requested view-window. 'rsiz' is used to identify the horizontal and vertical extents of the spatial region associated with the requested view-window.

6.80.3 JPIPKAK -approach

The JPIPKAK driver uses an approach that was first demonstrated here, J2KViewer, by Juan Pablo Garcia Ortiz of separating the communication layer (socket / http) from the Kakadu kdu_cache object. Separating the communication layer from the data object is desirable since it allows the use of optimized http client libraries such as libcurl, Apache HttpClient (note that jportiz used a plain Java socket) and allows SSL communication between the client and server.

Kakadu's implementation of client communication with a JPIP server uses a socket, and this socket connection holds the state for this client session. A client session with Kakadu can be recreated using the JPIP cache operations between client and server, but no use of traditional HTTP cookies is supported since JPIP is neutral to the transport layer.

The JPIPKAK driver is written using a HTTP client library with the Kakadu cache object and supports optimized communication with a JPIP server (which may or may not support HTTP sessions) and the high performance of the kakadu kdu_region_decompressor.



6.80.4 JPIPKAK - implementation

The implementation supports the GDAL C++ and C API, and provides an initial SWIG wrapper for this driver with a Java ImageIO example (**TODO** - qGIS Example).

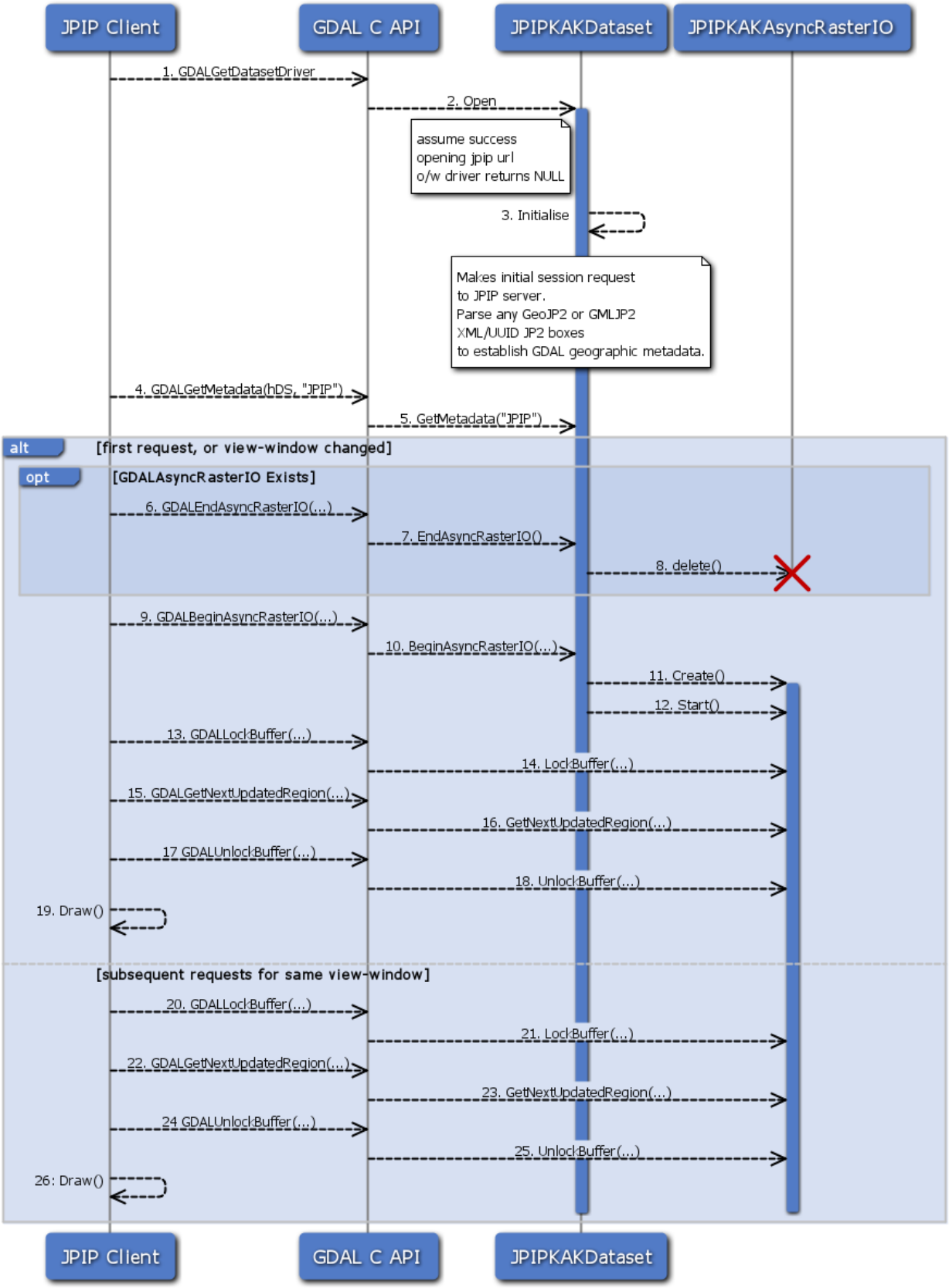
The driver uses a simple threading model to support requesting reads of the data and remote fetching. This threading model supports two separate client windows, with just one connection to the server. Requests to the server are multiplexed to utilize available bandwidth efficiently. The client identifies these windows by using "0" (low) or "1" (high) values to a "PRIORITY" metadata request option.

Note: SSL support

If the client is built with support for SSL, then driver determines whether to use SSL if the request is a jpips:// protocol as opposed to jpip:// . Note that the driver does not verify server certificates using the Curl certificate bundle and is currently set to accept all SSL server certificates.

Note: libCurl

JPIP sets client/server values using HTTP headers, modifications have been made to the GDAL HTTP portability library to support this.



1. GDALGetDatasetDriver

Fetch the driver to which this dataset relates.

2. Open

   If the filename contained in the `GDALOpenInfo` object has a case insensitive URI scheme of JPIP or JPIPS the `JPIPKAKDataset` is created and initialized, otherwise NULL is returned.

3. Initialize

   Initialization involves making an initial connection to the JPIP Server to establish a session and to retrieve the initial metadata about the image (ref. *JPIP Sequence Diagram*).

   If the connection fails, the function returns false and the `Open` function returns NULL indicating that opening the dataset with this driver failed.

   If the connection is successful, then subsequent requests to the JPIP server are made to retrieve all the available metadata about the image. Metadata items are set using the `GDALMajorObject->SetMetadataItem` in the "JPIP" domain.

   If the metadata returned from the server includes GeoJP2 UUID box, or a GMLJP2 XML box then this metadata is parsed and sets the geographic metadata of this dataset.

4. GDALGetMetadata

   C API to `JPIPKAKDataset->GetMetadata`

5. GetMetadata

   returns metadata for the "JPIP" domain, keys are "JPIP_NQUALITYLAYERS", "JPIP_NRESOLUTIONLEVELS", "JPIP_NCOMPS" and "JPIP_SPRECISION"

6. GDALEndAsyncRasterIO

   If the asynchronous raster IO is active and not required, the C API calls `JPIPKAKDataset->EndAsyncRasterIO`

7. EndAsyncRasterIO

   The JPIPKAKAsyncRasterIO object is deleted

8. delete

9. GDALBeginAsyncRasterIO

   C API to `JPIPKAKDataset->BeginAsyncRasterIO`

10. BeginAsyncRasterIO

    The client has set the requested view window at 1:1 and have optionally set the discard level, quality layers and thread priority metadata items.

11. Create

    Creates a JPIPKAKAsyncRasterIO Object

12. Start

    Configures the kakadu machinery and starts a background thread (if not already running) to communicate to the server the current view window request. The background thread results in the `kdu_cache` object being updated until the JPIP server sends an "End Of Response" (EOR) message for the current view window request.

13. GDALLockBuffer

    C API to LockBuffer

14. LockBuffer

    Not implemented in `JPIPKAKAsyncRasterIO`, a lock is acquired in `JPIPKAKAsyncRasterIO->GetNextUpdatedRegion`

15. GDALGetNextUpdatedRegion

    C API to GetNextUpdatedRegion

16. GetNextUpdatedRegion

    The function decompresses the available data to generate an image (according to the dataset buffer type set in `JPIPKAKDataset->BeginAsyncRasterIO`) The window width, height (at the requested discard level) decompressed is returned in the region pointer and can be rendered by the client. The status of the rendering operation is one of `GARIO_PENDING, GARIO_UPDATE, GARIO_ERROR, GARIO_COMPLETE` from the `GDALAsyncStatusType` structure. `GARIO_UPDATE, GARIO_PENDING` require more reads of GetNextUpdatedRegion to get the full image data, this is the progressive rendering of JPIP. `GARIO_COMPLETE` indicates the window is complete.

    `GDALAsyncStatusType` is a structure used by`GetNextUpdatedRegion` to indicate whether the function should be called again when either kakadu has more data in its cache to decompress, or the server has not sent an End Of Response (EOR) message to indicate the request window is complete.

    The region passed into this function is passed by reference, and the caller can read this region when the result returns to find the region that has been decompressed. The image data is packed into the buffer, e.g. RGB if the region requested has 3 components.

17. GDALUnlockBuffer

    C Api to UnlockBuffer

18. UnlockBuffer

    Not implemented in `JPIPKAKAsyncRasterIO`, a lock is acquired in `JPIPKAKAsyncRasterIO->GetNextUpdatedRegion`

19. Draw

    Client renders image data

20. *GDALLockBuffer*
21. *LockBuffer*
22. *GDALGetNextUpdatedRegion*
23. *GetNextUpdatedRegion*
24. *GDALUnlockBuffer*
25. *UnlockBuffer*
26. *Draw*

6.80.5 JPIPKAK - installation requirements

- Libcurl 7.9.4
- OpenSSL 0.9.8K (if SSL is required, a JPIPS connection)
- Kakadu (tested with v5.2.6 and v6)

Currently only a Windows makefile is provided, however this should compile on Linux as well as there are no Windows dependencies.

6.80.6 See Also

- JPEG 2000 Interactive Protocol (Part 9 – JPIP)
- http://www.opengeospatial.org/standards/gmljp2
- Kakadu Software
- IAS demo (example JPIP(S) streams)

6.80.7 NOTES

Driver originally developed by ITT VIS and donated to GDAL to enable SSL enabled JPIP client streaming of remote JPEG 2000 datasets.

6.81 KEA

Driver short name

KEA

Build dependencies

libkea and libhdf5 libraries

GDAL can read, create and update files in the KEA format, through the libkea library.

KEA is an image file format, named after the New Zealand bird, that provides a full implementation of the GDAL data model and is implemented within a HDF5 file. A software library, libkea, is used to access the file format. The format has comparable performance with existing formats while producing smaller file sizes and is already within active use for a number of projects within Landcare Research, New Zealand, and the wider community.

The KEA format supports the following features of the GDAL data model:

- Multiple-band support, with possible different datatypes. Bands can be added to an existing dataset with AddBand() API
- Image blocking support
- Reading, creation and update of data of image blocks
- Affine geotransform, WKT projection, GCP
- Metadata at dataset and band level
- Per-band description
- Per-band nodata and color interpretation
- Per-band color table
- Per-band RAT (Raster Attribute Table) of arbitrary size
- Internal overviews and mask bands

6.81.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Added in version 3.0:

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.81.2 Creation options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following creation options are available. Some are rather esoteric and should rarely be specified, unless the user has good knowledge of the working of the underlying HDF5 format.

- **IMAGEBLOCKSIZE=<integer>:** Defaults to `256`. The size of each block for image data.
- **ATTBLOCKSIZE=<integer>:** Defaults to `1000`. The size of each block for attribute data.
- **MDC_NELMTS=<integer>:** Defaults to `0`. Number of elements in the meta data cache. Defaults to 0. See the Data caching page of HDF5 documentation.
- **RDCC_NELMTS=<integer>:** Defaults to `512`. Number of elements in the raw data chunk cache. See the Data caching page of HDF5 documentation.
- **RDCC_NBYTES=<bytes>:** Defaults to `1048576`. Total size of the raw data chunk cache, in bytes. See the Data caching page of HDF5 documentation.
- **RDCC_W0=<floating_point_value between 0 and 1>:** Defaults to `0.75`. Preemption policy. See the Data caching page of HDF5 documentation.
- **SIEVE_BUF=<integer>:** Defaults to `65536`. Sets the maximum size of the data sieve buffer. See H5Pset_sieve_buf_size() documentation
- **META_BLOCKSIZE=<integer>:** Defaults to `2048`. Sets the minimum size of metadata block allocations. See H5Pset_meta_block_size() documentation
- **DEFLATE=[0-9]:** Defaults to `1`. Compression level between 0 (no compression) to 9 (max compression).
- **THEMATIC=[YES/NO]:** Defaults to `NO`. If YES then all bands are set to thematic.

6.81.3 See Also

- libkea GitHub repository
- The KEAimage file format, by Peter Bunting and Sam Gillingham, published in Computers&Geosciences
- *HDF5 driver page*

6.82 KMLSuperoverlay -- KMLSuperoverlay

Driver short name

KMLSuperoverlay

Driver built-in by default

This driver is built-in by default

NOTE: Implemented as kmlsuperoverlaydataset.cpp.

6.82.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.83 KRO -- KOLOR Raw format

Driver short name

KRO

Driver built-in by default

This driver is built-in by default

Supported for read access, update and creation. This format is a binary raw format, that supports data of several depths (8 bit, unsigned integer 16 bit and floating point 32 bit) and with several band number (3 or 4 typically, for RGB and RGBA). There is no file size limit, except the limitation of the file system.

Specification of the format

NOTE: Implemented as krodataset.cpp.

6.83.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.84 KTX2

Added in version 3.6.

Driver short name

KTX2

Build dependencies

Basis Universal

From https://github.com/BinomialLLC/basis_universal,

> Basis Universal is a "supercompressed" GPU texture data interchange system that supports two highly compressed intermediate file formats (.basis or the .KTX2 open standard from the Khronos Group) that can be quickly transcoded to a very wide variety of GPU compressed and uncompressed pixel formats.

This driver handles textures with the .ktx2 extension. For .basic, refer to the *BASISU -- Basis Universal* driver. Note that this driver does *not* handle KTX(1) files.

Note that while the file format supports direct transcoding to other GPU pixel formats, this GDAL driver supports only conversion between uncompressed RGB(A) data and Basis Universal textures.

When a file is made of several images, they are exposed as subdataset, with the *KTX2:filename:layer_idx:face_idx* syntax.

Mipmaps levels are exposed as GDAL overviews.

6.84.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.84.2 Creation options

The format supports 2 modes: a high quality mode which is internally based off the UASTC compressed texture format, and the original lower quality mode which is based off a subset of ETC1 called "ETC1S". The default is ETC1S. Only input of type Byte and with 1 (grey), 2 (grey+alpha), 3 (RGB) or 4 (RGB + alpha) bands is supported. Refer to https://github.com/BinomialLLC/basis_universal for more details on those modes and their options.

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following creation options are available:

- **COMPRESSION=[ETC1S/UASTC]:** Defaults to `ETC1S`.
- **UASTC_SUPER_COMPRESSION=[ZSTD/NONE]:** Defaults to `ZSTD`. "Super" compression to apply. Only valid when *COMPRESSION=UASTC*.
- **UASTC_LEVEL=[0/1/2/3/4]:** Defaults to `2`. The higher value, the higher the quality but the slower computing time. 4 is impractically slow. Only valid when *COMPRESSION=UASTC*.
- **UASTC_RDO_LEVEL=<float>:** Defaults to `1`. Rate distortion optimization level. The lower value, the higher the quality, but the larger the file size. Usual range is [0.2,3]. Only valid when *COMPRESSION=UASTC*.
- **ETC1S_LEVEL=[0/1/2/3/4/5/6]:** Defaults to `1`. The higher value, the higher the quality but the slower computing time. Only valid when *COMPRESSION=ETC1S*.
- **ETC1S_QUALITY_LEVEL=1-255:** Defaults to `128`. The higher value, the higher the quality, but the larger the file size. Only valid when *COMPRESSION=ETC1S*.
- **ETC1S_MAX_ENDPOINTS_CLUSTERS=1-16128:** Maximum number of endpoint clusters. When set, *ETC1S_MAX_SELECTOR_CLUSTERS* must also be set. Mutually exclusive with *ETC1S_QUALITY_LEVEL*. Only valid when *COMPRESSION=ETC1S*.
- **ETC1S_MAX_SELECTOR_CLUSTERS=1-16128.:** Maximum number of selector clusters. When set, *ETC1S_MAX_ENDPOINTS_CLUSTERS* must also be set. Mutually exclusive with *ETC1S_QUALITY_LEVEL*. Only valid when *COMPRESSION=ETC1S*.
- **NUM_THREADS=<integer>:** Defaults to the maximum number of virtual CPUs available. Can also be controlled with the *GDAL_NUM_THREADS* configuration option
- **MIPMAP=[YES/NO]:** Defaults to `NO`. Whether to enable MIPMAP generation.
- **COLORSPACE=[PERCEPTUAL_SRGB/LINEAR]:** Defaults to `PERCEPTUAL_SRGB`. For non-photometric input, use LINEAR to avoid unnecessary artifacts.

6.84.3 Build instructions

Building basisu as a library requires currently building the *cmake* branch of the https://github.com/rouault/basis_universal/tree/cmake fork.

```
git clone -b cmake https://github.com/rouault/basis_universal
cd basis_universal
mkdir build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=/path/to/install-basisu -DCMAKE_BUILD_TYPE=Release -
↪DBUILD_SHARED_LIBS=ON
cmake --build . --config Release --target install
```

Once basisu is built, GDAL CMake options must be configured by pointing the basisu install prefix in the `CMAKE_PREFIX_PATH` variable or `basisu_ROOT` variable.

6.85 LAN -- Erdas 7.x .LAN and .GIS

Driver short name

LAN

Driver built-in by default

This driver is built-in by default

GDAL supports reading Erdas 7.x .LAN and .GIS raster files. Currently 4bit, 8bit and 16bit pixel data types are supported for reading.

GDAL does read the map extents (geotransform) from LAN/GIS files, and attempts to read the coordinate system information. However, this format of file does not include complete coordinate system information, so for state plane and UTM coordinate systems a LOCAL_CS definition is returned with valid linear units but no other meaningful information.

The .TRL, .PRO and worldfiles are ignored at this time.

NOTE: Implemented as landataset.cpp

Development of this driver was financially supported by Kevin Flanders of (PeopleGIS).

6.85.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.86 L1B -- NOAA Polar Orbiter Level 1b Data Set (AVHRR)

Driver short name

L1B

Driver built-in by default

This driver is built-in by default

GDAL supports NOAA Polar Orbiter Level 1b Data Set format for reading. Now it can read NOAA-9(F) -- NOAA-17(M) datasets. NOTE: only AVHRR instrument supported now, if you want read data from other instruments, write to me (Andrey Kiselev, dron@ak4719.spb.edu). AVHRR LAC/HRPT (1 km resolution) and GAC (4 km resolution) should be processed correctly.

6.86.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.86.2 Georeference

Note that GDAL simple affine georeference model is completely unsuitable for the NOAA data. So you should not rely on it. It is recommended to use the thin plate spline warper (tps). Automatic image rectification can be done with ground control points (GCPs) from the input file.

NOAA stores 51 GCPs per scanline both in the LAC and GAC datasets. In fact you may get less than 51 GCPs, especially at end of scanlines. Another approach to rectification is manual selection of the GCPs using external source of georeference information.

A high density of GCPs will be reported, unless the L1B_HIGH_GCP_DENSITY configuration option is set to NO.

Precision of the GCPs determination depends from the satellite type. In the NOAA-9 -- NOAA-14 datasets geographic coordinates of the GCPs stored in integer values as a 128th of a degree. So we can't determine positions more precise than 1/128=0.0078125 of degree (~28"). In NOAA-15 -- NOAA-17 datasets we have much more precise positions, they are stored as 10000th of degree.

The GCPs will also be reported as a geolocation array, with Lagrangian interpolation of the 51 GCPs per scanline to the number of pixels per scanline width.

Image will be always returned with most northern scanline located at the top of image. If you want determine actual direction of the satellite moving you should look at **LOCATION** metadata record.

6.86.3 Data

In case of NOAA-10 in channel 5 you will get repeated channel 4 data.

AVHRR/3 instrument (NOAA-15 -- NOAA-17) is a six channel radiometer, but only five channels are transmitted to the ground at any given time. Channels 3A and 3B cannot operate simultaneously. Look at channel description field reported by `gdalinfo` to determine what kind of channel contained in processed file.

6.86.4 Metadata

Several parameters, obtained from the dataset stored as metadata records.

Metadata records:

- **SATELLITE**: Satellite name
- **DATA_TYPE**: Type of the data, stored in the Level 1b dataset (AVHRR HRPT/LAC/GAC).
- **REVOLUTION**: Orbit number. Note that it can be 1 to 2 off the correct orbit number (according to documentation).
- **SOURCE**: Receiving station name.
- **PROCESSING_CENTER**: Name of data processing center.
- **START**: Time of first scanline acquisition (year, day of year, millisecond of day).
- **STOP**: Time of last scanline acquisition (year, day of year, millisecond of day).
- **LOCATION**: AVHRR Earth location indication. Will be **Ascending** when satellite moves from low latitudes to high latitudes and **Descending** in other case.

Most metadata records can be written to a .CSV file when the L1B_FETCH_METADATA configuration file is set to YES. By default, the filename will be called "[l1b_dataset_name]_metadata.csv", and located in the same directory as the L1B dataset. By defining the L1B_METADATA_DIRECTORY configuration option, it is possible to create that file in another directory. The documentation to interpret those metadata is PODUG 3.1 for NOAA <=14 and KLM 8.3.1.3.3.1 for NOAA >=15.

6.86.5 Subdatasets

NOAA <=14 datasets advertise a L1B_SOLAR_ZENITH_ANGLES:"l1b_dataset_name" subdataset that contains a maximum of 51 solar zenith angles for each scanline (beginning at sample 5 with a step of 8 samples for GAC data, beginning at sample 25 with a step of 40 samples for HRPT/LAC/FRAC data).

NOAA >=15 datasets advertise a L1B_ANGLES:"l1b_dataset_name" subdataset that contains 3 bands (solar zenith angles, satellite zenith angles and relative azimuth angles) with 51 values for each scanline (beginning at sample 5 with a step of 8 samples for GAC data, beginning at sample 25 with a step of 40 samples for HRPT/LAC/FRAC data).

NOAA >=15 datasets advertise a L1B_CLOUDS:"l1b_dataset_name" subdataset that contains a band of same dimensions as bands of the main L1B dataset. The values of each pixel are 0 = unknown; 1 = clear; 2 = cloudy; 3 = partly cloudy.

6.86.6 Nodata mask

NOAA >=15 datasets that report in their header to have missing scan lines will expose a per-dataset mask band (following rfc-15) to indicate such scan lines.

6.86.7 See Also

- Implemented as l1bdataset.cpp.
- NOAA Polar Orbiter Level 1b Data Set documented in the ``POD User's Guide'' (TIROS-N -- NOAA-14 satellites) and in the ``NOAA KLM User's Guide'' (NOAA-15 -- NOAA-16 satellites). You can find this manuals at NOAA Technical Documentation Introduction Page
- There are a great variety of L1B datasets, sometimes with variations in header locations that are not documented in the official NOAA documentation. In case a dataset is not recognized by the GDAL L1B driver, the pytroll package might be able to recognize it.
- Excellent and complete review contained in the printed book ``The Advanced Very High Resolution Radiometer (AVHRR)'' by Arthur P. Cracknell, Taylor and Francis Ltd., 1997, ISBN 0-7484-0209-8.
- NOAA data can be downloaded from the Comprehensive Large Array-data Stewardship System (CLASS) (former SAA). Actually it is only source of Level 1b datasets for me, so my implementation tested with that files only.
- NOAA spacecrafts status page

6.87 LCP -- FARSITE v.4 LCP Format

Driver short name

LCP

Driver built-in by default

This driver is built-in by default

FARSITE v. 4 landscape file (LCP) is a multi-band raster format used by wildland fire behavior and fire effect simulation models such as FlamMap (firelab.org). The bands of an LCP file store data describing terrain, tree canopy, and surface fuel. The LANDFIRE Program provides geospatial data in LCP format to support fire simulation modeling.

An LCP file (.lcp) is basically a raw format with a 7,316-byte header described below. The data type for all bands is 16-bit signed integer. Bands are interleaved by pixel. Five bands are required: elevation, slope, aspect, fuel model, and tree canopy cover. Crown fuel bands (canopy height, canopy base height, canopy bulk density), and surface fuel bands (duff, coarse woody debris) are optional. The LCP driver reports several band-level metadata items described below. The band names used are: ELEVATION, SLOPE, ASPECT, FUEL_MODEL, CANOPY_COV, CANOPY_HT, CBH, CBD, DUFF and CWD.

The LCP driver reads the linear unit, cell size, and extent, but the .lcp file does not specify a projection internally. UTM projections are typical, but other projections are possible. The driver will look for a file with .prj extension and read the coordinate system in ESRI pseudo-OGC WKT format.

LCP does not support an inherent no data value, but -9999 is used by convention to mark pixels that are not valid data.

6.87.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.87.2 Metadata

The GDAL LCP driver reports dataset- and band-level metadata:

6.87.2.1 Dataset

LATITUDE: Latitude of the dataset, negative for southern hemisphere
LINEAR_UNIT: Feet or meters
DESCRIPTION: LCP file description

6.87.2.2 Band

<band>_UNIT or <band>_OPTION: units or options code for the band
<band>_UNIT_NAME or <band>_OPTION_DESC: descriptive name of units/options
<band>_MIN: minimum value
<band>_MAX: maximum value
<band>_NUM_CLASSES: number of classes, -1 if > 100
<band>_VALUES: comma-delimited list of class values (fuel model band only)
<band>_FILE: original input raster file name for the band

Note: The LCP driver derives from the RawDataset helper class declared in *frmts/raw*. It should be implemented as lcpdataset.cpp.

6.87.3 Creation Options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The LCP driver supports CreateCopy() and metadata values can be set via creation options. Below is a list of options with default values listed first.

- **ELEVATION_UNIT=[METERS/FEET]:** Defaults to `METERS`. Vertical unit for elevation band.
- **SLOPE_UNIT=[DEGREES/PERCENT]:** Defaults to `DEGREES`.
- **ASPECT_UNIT=[AZIMUTH_DEGREES/GRASS_CATEGORIES/GRASS_DEGREES]:** Defaults to `AZIMUTH_DEGREES`.
- **FUEL_MODEL_OPTION=[NO_CUSTOM_AND_NO_FILE/CUSTOM_AND_NO_FILE/NO_CUSTOM_AND_FILE/CUSTOM_AND_FILE]:** Defaults to `NO_CUSTOM_AND_NO_FILE`. Specify whether or not custom fuel models are used, and if a custom fuel model file is present.
- **CANOPY_COV_UNIT=[PERCENT/CATEGORIES]:** Defaults to `PERCENT`.
- **CANOPY_HT_UNIT=[METERS_X_10/FEET/METERS/FEET_X_10]:** Defaults to `METERS_X_10`.
- **CBH_UNIT=[METERS_X_10/METERS/FEET/FEET_X_10]:** Defaults to `METERS_X_10`.

- **CBD_UNIT=[KG_PER_CUBIC_METER_X_100/POUND_PER_CUBIC_FOOT/ KG_PER_CUBIC_METER/POUND_PER_CUBIC_FOOT_X_1000/TONS_PER_ACRE_X_100]:** Defaults to `KG_PER_CUBIC_METER_X_100`.
- **DUFF_UNIT=[MG_PER_HECTARE_X_10/TONS_PER_ACRE_X_10]:** Defaults to `MG_PER_HECTARE_X_10`.
- **CALCULATE_STATS=[YES/NO]:** Defaults to `YES`. Calculate and write the min/max for each band and write the appropriate flags and values in the header. This is mostly a legacy feature used for creating legends.
- **CLASSIFY_DATA=[YES/NO]:** Defaults to `YES`. Classify the data into 100 unique values or less and write and write the appropriate flags and values in the header. This is mostly a legacy feature used for creating legends.
- **LINEAR_UNIT=[SET_FROM_SRS/METER/FOOT/KILOMETER]:** Defaults to `SET_FROM_SRS`. Set the linear unit, overriding (if it can be calculated) the value in the associated spatial reference. If no spatial reference is available, it defaults to METER.
- **LATITUDE=-90-90:** Override the latitude from the spatial reference. If no spatial reference is available, this should be set, otherwise creation will fail.
- **DESCRIPTION=value:** A short description(less than 512 characters) of the dataset

Creation options that are units of linear measure are fairly lenient. METERS=METER and FOOT=FEET for the most part.

Note: CreateCopy does not scale or change any data. By setting the units for various bands, it is assumed that the values are in the specified units.

LCP header format:

Start byte	No. of bytes	Format	Name	Description
0	4	long	crown fuels	20 if no crown fuels, 21 if crown fuels exist (crown fuels = canopy height, c
4	4	long	ground fuels	20 if no ground fuels, 21 if ground fuels exist (ground fuels = duff loading,
8	4	long	latitude	latitude (negative for southern hemisphere)
12	8	double	loeast	offset to preserve coordinate precision (legacy from 16-bit OS days)
20	8	double	hieast	offset to preserve coordinate precision (legacy from 16-bit OS days)
28	8	double	lonorth	offset to preserve coordinate precision (legacy from 16-bit OS days)
36	8	double	hinorth	offset to preserve coordinate precision (legacy from 16-bit OS days)
44	4	long	loelev	minimum elevation
48	4	long	hielev	maximum elevation
52	4	long	numelev	number of elevation classes, -1 if > 100
56	400	long	elevation values	list of elevation values as longs
456	4	long	loslope	minimum slope
460	4	long	hislope	maximum slope
464	4	long	numslope	number of slope classes, -1 if > 100
468	400	long	slope values	list of slope values as longs
868	4	long	loaspect	minimum aspect
872	4	long	hiaspect	maximum aspect
876	4	long	numaspects	number of aspect classes, -1 if > 100
880	400	long	aspect values	list of aspect values as longs
1280	4	long	lofuel	minimum fuel model value
1284	4	long	hifuel	maximum fuel model value
1288	4	long	numfuel	number of fuel models -1 if > 100
1292	400	long	fuel values	list of fuel model values as longs
1692	4	long	locover	minimum canopy cover
1696	4	long	hicover	maximum canopy cover
1700	4	long	numcover	number of canopy cover classes, -1 if > 100

Table 6 – continued fr

1704	400	long	cover values	list of canopy cover values as longs
2104	4	long	loheight	minimum canopy height
2108	4	long	hiheight	maximum canopy height
2112	4	long	numheight	number of canopy height classes, -1 if > 100
2116	400	long	height values	list of canopy height values as longs
2516	4	long	lobase	minimum canopy base height
2520	4	long	hibase	maximum canopy base height
2524	4	long	numbase	number of canopy base height classes, -1 if > 100
2528	400	long	base values	list of canopy base height values as longs
2928	4	long	lodensity	minimum canopy bulk density
2932	4	long	hidensity	maximum canopy bulk density
2936	4	long	numdensity	number of canopy bulk density classes, -1 if >100
2940	400	long	density values	list of canopy bulk density values as longs
3340	4	long	loduff	minimum duff
3344	4	long	hiduff	maximum duff
3348	4	long	numduff	number of duff classes, -1 if > 100
3352	400	long	duff values	list of duff values as longs
3752	4	long	lowoody	minimum coarse woody
3756	4	long	hiwoody	maximum coarse woody
3760	4	long	numwoodies	number of coarse woody classes, -1 if > 100
3764	400	long	woody values	list of coarse woody values as longs
4164	4	long	numeast	number of raster columns
4168	4	long	numnorth	number of raster rows
4172	8	double	EastUtm	max X
4180	8	double	WestUtm	min X
4188	8	double	NorthUtm	max Y
4196	8	double	SouthUtm	min Y
4204	4	long	GridUnits	linear unit: 0 = meters, 1 = feet, 2 = kilometers
4208	8	double	XResol	cell size width in GridUnits
4216	8	double	YResol	cell size height in GridUnits
4224	2	short	EUnits	elevation units: 0 = meters, 1 = feet
4226	2	short	SUnits	slope units: 0 = degrees, 1 = percent
4228	2	short	AUnits	aspect units: 0 = Grass categories, 1 = Grass degrees, 2 = azimuth degrees
4230	2	short	FOptions	fuel model options: 0 = no custom models AND no conversion file, 1 = cus
4232	2	short	CUnits	canopy cover units: 0 = categories (0-4), 1 = percent
4234	2	short	HUnits	canopy height units: 1 = meters, 2 = feet, 3 = m x 10, 4 = ft x 10
4236	2	short	BUnits	canopy base height units: 1 = meters, 2 = feet, 3 = m x 10, 4 = ft x 10
4238	2	short	PUnits	canopy bulk density units: 1 = kg/m^3, 2 = lb/ft^3, 3 = kg/m^3 x 100, 4 = l
4240	2	short	DUnits	duff units: 1 = Mg/ha x 10, 2 = t/ac x 10
4242	2	short	WOptions	coarse woody options (1 if coarse woody band is present)
4244	256	char[]	ElevFile	elevation file name
4500	256	char[]	SlopeFile	slope file name
4756	256	char[]	AspectFile	aspect file name
5012	256	char[]	FuelFile	fuel model file name
5268	256	char[]	CoverFile	canopy cover file name
5524	256	char[]	HeightFile	canopy height file name
5780	256	char[]	BaseFile	canopy base file name
6036	256	char[]	DensityFile	canopy bulk density file name
6292	256	char[]	DuffFile	duff file name
6548	256	char[]	WoodyFile	coarse woody file name
6804	512	char[]	Description	LCP file description

Chris Toney, 2009-02-14

6.88 Leveller -- Daylon Leveller Heightfield

Driver short name

Leveller

Driver built-in by default

This driver is built-in by default

Leveller heightfields store 32-bit elevation values. Format versions 4 through 12 are supported with various caveats (see below). The file extension for Leveller heightfields is "TER" (which is the same as Terragen, but the driver only recognizes Leveller files).

Blocks are organized as pixel-high scanlines (rows), with the first scanline at the top (north) edge of the DEM, and adjacent pixels on each line increasing from left to right (west to east).

The band type is always Float32, even though format versions 4 and 5 physically use 16.16 fixed-point. The driver auto-converts them to floating-point.

6.88.1 Driver capabilities

Supports CreateCopy()

This driver supports the *`GDALDriver::CreateCopy()`* operation

Supports Create()

This driver supports the *`GDALDriver::Create()`* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.88.2 Reading

`dataset::GetProjectionRef()` will return only a local coordinate system for file versions 4 through 6.

`dataset::GetGeoTransform()` returns a simple world scaling with a centered origin for formats 4 through 6. For versions 7 and higher, it returns a real-world transform except for rotations. The identity transform is not considered an error condition; Leveller documents often use them.

`band::GetUnitType()` will report the measurement units used by the file instead of converting unusual types to meters. A list of unit types is in the `levellerdataset.cpp` module.

`band::GetScale()` and `band::GetOffset()` will return the physical-to-logical (i.e., raw to real-world) transform for the elevation data.

6.88.3 Writing

The `dataset::Create()` call is supported, but for version 7 files only.

`band::SetUnitType()` can be set to any of the unit types listed in the `levellerdataset.cpp` module.

`dataset::SetGeoTransform()` should not include rotation data.

As with the Terragen driver, the `MINUSERPIXELVALUE` option must be specified. This lets the driver correctly map from logical (real-world) elevations to physical elevations.

Header information is written out on the first call to `band::IWriteBlock`.

6.88.4 See Also:

- Implemented as levellerdataset.cpp.
- Visit Daylon Graphics for the Leveller SDK, which documents the Leveller format.

6.89 LIBERTIFF -- GeoTIFF File Format

Added in version 3.11.

Driver short name

LIBERTIFF

Driver built-in by default

This driver is built-in by default

This driver is a natively thread-safe alternative to the default *GTiff -- GeoTIFF File Format* driver. Note that the driver is read-only.

The driver is registered after the GTiff one. Consequently one must explicitly specify `LIBERTIFF` in the allowed drivers of the *`GDALOpenEx()`*, or with the `-if` option of command line utilities, to use it.

The driver supports the following compression methods: LZW, Deflate, PackBits, LZMA, ZSTD, LERC, JPEG, WEBP and JPEGXL. The driver supports only BitsPerSample values of 1, 8, 16, 32, 64.

The driver does *not* read any side-car file: `.aux.xml`, `.ovr`, `.msk`, `.imd`, etc.

The driver mostly by-passes the GDAL raster block cache, and only caches (per-thread) the last tile or strip it has read. Read patterns must be adapted accordingly, to avoid repeated data acquisition from storage and decompression.

6.89.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.89.2 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). This driver supports the following open options:

NUM_THREADS=[<number_of_threads>/ALL_CPUS]: This option also enables multi-threaded decoding when RasterIO() requests intersect several tiles/strips. The *`GDAL_NUM_THREADS`* configuration option can also be used as an alternative to setting the open option.

6.90 LOSLAS -- NADCON .los/.las Datum Grid Shift

Driver short name

LOSLAS

Driver built-in by default

This driver is built-in by default

Starting with GDAL 3.1, can also open geoid models of extension .geo such as https://geodesy.noaa.gov/GEOID/MEXICO97/

NOTE: Implemented as loslasdataset.cpp.

6.90.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.91 MAP -- OziExplorer .MAP

Driver short name

MAP

Driver built-in by default

This driver is built-in by default

OziExplorer MAP files.

6.91.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.91.2 See Also:

- Implemented as mapdataset.cpp.
- OziExplorer Map File Format

6.92 MRF -- Meta Raster Format

Driver short name

MRF

Driver built-in by default

This driver is built-in by default

Access to a indexed heap of regular tiles (blocks). Controlled by an xml file, usually organized as a pyramid of overviews, with level zero being the full resolution image. None, PNG, JPEG, ZLIB tile packing are implemented

For file creation options, see "gdalinfo --format MRF"

6.92.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.92.2 Links

- MRF User Guide
- MRF Specification
- Source repository nasa-gibs/mrf

6.93 MBTiles

Driver short name

MBTiles

Build dependencies

libsqlite3

The MBTiles driver allows reading, creating and modifying rasters in the MBTiles format, which is a specification for storing tiled map data in SQLite databases.

The MBTiles driver has read and write support for MBTiles vector datasets. For standalone Mapbox Vector Tile files or set of MVT files, see the *MVT* driver. Note: vector write support requires GDAL to be built with GEOS.

GDAL/OGR must be compiled with OGR SQLite driver support, and JPEG and PNG drivers.

The SRS is always the Pseudo-Mercator (a.k.a Google Mercator) projection, EPSG:3857.

The driver will open a dataset as RGBA. For previous versions, the driver will try to determine the number of bands by probing the content of one tile. It is possible to alter this behavior by defining the *MBTILES_BAND_COUNT* configuration

option (or the *BAND_COUNT* open option) to the number of bands. The values supported are 1, 2, 3 or 4. Four band (Red,Green,Blue,Alpha) dataset gives the maximum compatibility with the various encodings of tiles that can be stored.

The driver will use the 'bounds' metadata in the metadata table and do necessary tile clipping, if needed, to respect that extent. However that information being optional, if omitted, the driver will use the extent of the tiles at the maximum zoom level. The user can also specify the *USE_BOUNDS=NO* open option to force the use of the actual extent of tiles at the maximum zoom level. Or it can specify any of MINX/MINY/MAXX/MAXY to have a custom extent.

The driver can retrieve pixel attributes encoded according to the UTFGrid specification available in some MBTiles files. They can be obtained with the gdallocationinfo utility, or with a GetMetadataItem("Pixel_iCol_iLine", "LocationInfo") call on a band object.

6.93.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.93.2 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). The following configuration option is available:

- **MBTILES_BAND_COUNT=value:** Equivalent of *BAND_COUNT* open option.

6.93.3 Opening options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are available:

- Raster and vector:
  - **ZOOM_LEVEL=value:** Integer value between 0 and the maximum filled in the *tiles* table. By default, the driver will select the maximum zoom level, such as at least one tile at that zoom level is found in the 'tiles' table.
  - **USE_BOUNDS=[YES/NO]:** Defaults to `YES`. Whether to use the 'bounds' metadata, when available, to determine the AOI.

  - **MINX=value:** Minimum easting (in EPSG:3857) of the area of interest.
  - **MINY=value:** Minimum northing (in EPSG:3857) of the area of interest.
  - **MAXX=value:** Maximum easting (in EPSG:3857) of the area of interest.
  - **MAXY=value:** Maximum northing (in EPSG:3857) of the area of interest.
- Raster only:
  - **BAND_COUNT=[AUTO/1/2/3/4]:** Defaults to `AUTO`. Number of bands of the dataset exposed after opening. Some conversions will be done when possible and implemented, but this might fail in some cases, depending on the BAND_COUNT value and the number of bands of the tile.
  - **TILE_FORMAT=[PNG/PNG8/JPEG/WEBP]:** Defaults to `PNG`. Format used to store tiles. See *Tile formats* section. Only used in update mode.
  - **QUALITY=1-100:** Defaults to `75`. Quality setting for JPEG/WEBP compression. Only used in update mode.
  - **ZLEVEL=1-9:** Defaults to `6`. DEFLATE compression level for PNG tiles. Only used in update mode.
  - **DITHER=[YES/NO]:** Defaults to `NO`. Whether to use Floyd-Steinberg dithering (for *`TILE_FORMAT=PNG8`*). Only used in update mode.
- Vector only:
  - **CLIP=[YES/NO]:** Defaults to `YES`. Whether to clip geometries of vector features to tile extent.
  - **ZOOM_LEVEL_AUTO=[YES/NO]:** Defaults to `NO`. Whether to auto-select the zoom level for vector layers according to the spatial filter extent. Only for display purpose.

6.93.4 Raster creation issues

Depending of the number of bands of the input dataset and the tile format selected, the driver will do the necessary conversions to be compatible with the tile format. When using the CreateCopy() API (such as with *gdal_translate*), automatic reprojection of the input dataset to EPSG:3857 (Pseudo-Mercator) will be done, with selection of the appropriate zoom level.

Fully transparent tiles will not be written to the database, as allowed by the format.

The driver implements the Create() and IWriteBlock() methods, so that arbitrary writing of raster blocks is possible, enabling the direct use of MBTiles as the output dataset of utilities such as gdalwarp.

On creation, raster blocks can be written only if the geotransformation matrix has been set with SetGeoTransform() This is effectively needed to determine the zoom level of the full resolution dataset based on the pixel resolution, dataset and tile dimensions.

Technical/implementation note: in the general case, GDAL blocks do not exactly match a single MBTiles tile. In which case, each GDAL block will overlap four MBTiles tiles. This is easily handled on the read side, but on creation/update side, such configuration could cause numerous decompression/ recompression of tiles to be done, which might cause unnecessary quality loss when using lossy compression (JPEG). To avoid that, the driver will create a temporary database next to the main MBTiles file to store partial MBTiles tiles in a lossless (and uncompressed) way. Once a tile has received data for its four quadrants and for all the bands (or the dataset is closed or explicitly flushed with FlushCache()), those uncompressed tiles are definitely transferred to the MBTiles file with the appropriate compression. All of this is transparent to the user of GDAL API/utilities

6.93.4.1 Tile formats

MBTiles can store tiles in PNG, JPEG or WEBP (since 3.8). Support for those tile formats depend if the underlying drivers are available in GDAL. By default, GDAL will produce PNG tiles.

It is possible to select the tile format by setting the creation/open option TILE_FORMAT to one of PNG, PNG8, JPEG or WEBP. When using JPEG, the alpha channel will not be stored.

PNG8 can be selected to use 8-bit PNG with a color table up to 256 colors. On creation, an optimized color table is computed for each tile. The DITHER option can be set to YES to use Floyd/Steinberg dithering algorithm, which spreads the quantization error on neighbouring pixels for better rendering (note however than when zooming in, this can cause non desirable visual artifacts). Setting it to YES will generally cause less effective compression. Note that at that time, such an 8-bit PNG formulation is only used for fully opaque tiles, as the median-cut algorithm currently implemented to compute the optimal color table does not support alpha channel (even if PNG8 format would potentially allow color table with transparency). So when selecting PNG8, non fully opaque tiles will be stored as 32-bit PNG.

6.93.5 Vector creation issues

Tiles are generated with WebMercator (EPSG:3857) projection. It is possible to decide at which zoom level ranges a given layer is written. Several layers can be written but the driver has only write-once support for vector data. For writing several vector datasets into MBTiles file an intermediate format like GeoPackage must be used as a container so that all layers can be converted at the same time. Write-once support means also that existing vector layers can't be edited.

Part of the conversion is multi-threaded by default, using as many threads as there are cores. The number of threads used can be controlled with the *GDAL_NUM_THREADS* configuration option.

6.93.6 Creation options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following creation options are available:

- Raster and vector:
  - **NAME=value:** Tileset name, used to set the 'name' metadata item. If not specified, the basename of the filename will be used.
  - **DESCRIPTION=value:** A description of the layer, used to set the 'description' metadata item. If not specified, the basename of the filename will be used.
  - **TYPE=[overlay/baselayer]:** Defaults to `overlay`. The layer type, used to set the 'type' metadata item.
- Raster only:
  - **VERSION=value:** Defaults to `1.1`. The version of the tileset, as a plain number, used to set the 'version' metadata item.
  - **BLOCKSIZE=<integer>:** Defaults to `256`. Block/tile size in width and height in pixels. Maximum supported is 4096.
  - **TILE_FORMAT=[PNG/PNG8/JPEG/WEBP]:** Defaults to `PNG`. Format used to store tiles. See *Tile formats* section.
  - **QUALITY=1-100:** Defaults to `75`. Quality setting for JPEG/WEBP compression.
  - **ZLEVEL=1-9:** Defaults to `6`. DEFLATE compression level for PNG tiles.
  - **DITHER=[YES/NO]:** Defaults to `NO`. Whether to use Floyd-Steinberg dithering (for *TILE_FORMAT=PNG8*).

- **ZOOM_LEVEL_STRATEGY=[AUTO/LOWER/UPPER]:** Defaults to `AUTO`. Strategy to determine zoom level. LOWER will select the zoom level immediately below the theoretical computed non-integral zoom level, leading to subsampling. On the contrary, UPPER will select the immediately above zoom level, leading to oversampling. Defaults to AUTO which selects the closest zoom level.
- **RESAMPLING=[NEAREST/BILINEAR/CUBIC/CUBICSPLINE/LANCZOS/MODE/AVERAGE]:** Defaults to `BILINEAR`. Resampling algorithm.
- **WRITE_BOUNDS=[YES/NO]:** Defaults to `YES`. Whether to write the bounds 'metadata' item.
- **ELEVATION_TYPE=['']/['terrain-rgb']:** Defaults to `''`. Type of elevation encoding, suitable for direct generation of Mapbox Terrain-RGB tilesets. See Mapbox Terrain-RGB v1

• Vector only:

- **MINZOOM=<integer>:** Defaults to `0`. Minimum zoom level at which tiles are generated.
- **MAXZOOM=<integer>:** Defaults to `5`. Maximum zoom level at which tiles are generated. Maximum supported value is 22
- **CONF=value:** Layer configuration as a JSON serialized string.
- **SIMPLIFICATION=<float>:** Simplification factor for linear or polygonal geometries. The unit is the integer unit of tiles after quantification of geometry coordinates to tile coordinates. Applies to all zoom levels, unless *`SIMPLIFICATION_MAX_ZOOM`* is also defined.
- **SIMPLIFICATION_MAX_ZOOM=<float>:** Simplification factor for linear or polygonal geometries, that applies only for the maximum zoom level.
- **EXTENT=<integer>:** Defaults to `4096`. Number of units in a tile. The greater, the more accurate geometry coordinates (at the expense of tile byte size).
- **BUFFER=<integer>:** Number of units for geometry buffering. This value corresponds to a buffer around each side of a tile into which geometries are fetched and clipped. This is used for proper rendering of geometries that spread over tile boundaries by some rendering clients. Defaults to 80 if *`EXTENT=4096`*.
- **COMPRESS=[YES/NO]:** Defaults to `YES`. Whether to compress tiles with the Deflate/GZip algorithm. Defaults to YES. Should be left to YES for FORMAT=MBTILES.
- **TEMPORARY_DB=<filename>:** Filename with path for the temporary database used for tile generation. By default, this will be a file in the same directory as the output file/directory.
- **MAX_SIZE=<bytes>:** Defaults to `500000`. Maximum size of a tile in bytes (after compression). If a tile is greater than this threshold, features will be written with reduced precision, or discarded.
- **MAX_FEATURES=<integer>:** Defaults to `200000`. Maximum number of features per tile.
- **BOUNDS=<min_long,min_lat,max_long,max_lat>:** Override default value for bounds metadata item which is computed from the extent of features written.
- **CENTER=<long,lat,zoom_level>:** Override default value for center metadata item, which is the center of *`BOUNDS`* at minimum zoom level.

6.93.7 Layer configuration (vector)

The above mentioned CONF dataset creation option can be set to a string whose value is a JSON serialized document such as the below one:

```
{
    "boundaries_lod0": {
        "target_name": "boundaries",
        "description": "Country boundaries",
```

(continues on next page)

(continued from previous page)

```
        "minzoom": 0,
        "maxzoom": 2
    },
    "boundaries_lod1": {
        "target_name": "boundaries",
        "minzoom": 3,
        "maxzoom": 5
    }
}
```

boundaries_lod0 and *boundaries_lod1* are the name of the OGR layers that are created into the target MVT dataset. They are mapped to the MVT target layer *boundaries*.

It is also possible to get the same behavior with the below layer creation options, although that is not convenient in the ogr2ogr use case.

6.93.8 Layer creation options (vector)

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALDatasetCreateLayer()`* (C) or *`Dataset.CreateLayer`* (Python). The following layer creation options are available:

- **MINZOOM=<integer>:** Minimum zoom level at which tiles are generated. Defaults to the dataset creation option MINZOOM value.
- **MAXZOOM=<integer>:** Maximum zoom level at which tiles are generated. Defaults to the dataset creation option MAXZOOM value. Maximum supported value is 22
- **NAME=value:** Target layer name. Defaults to the layer name, but can be overridden so that several OGR layers map to a single target MVT layer. The typical use case is to have different OGR layers for mutually exclusive zoom level ranges.
- **DESCRIPTION=value:** A description of the layer.

6.93.9 Overviews (raster)

gdaladdo / BuildOverviews() can be used to compute overviews. Only power-of-two overview factors (2,4,8,16,...) are supported.

If more overview levels are specified than available, the extra ones are silently ignored.

Overviews can also be cleared with the -clean option of gdaladdo (or BuildOverviews() with nOverviews=0)

6.93.10 Vector tiles

The MBTiles driver can read MBTiles files containing vector tiles conforming to the Mapbox Vector Tile format (format=pbf).

The driver requires the 'metadata' table to contain a name='json' entry, that has a 'vector_layers' array describing layers and their schema. See *metadata.json*

Note: The driver will make no effort of stitching together geometries for features that overlap several tiles.

6.93.11 Examples:

- Accessing a remote MBTiles raster :

```
$ gdalinfo /vsicurl/http://a.tiles.mapbox.com/v3/kkaefer.iceland.mbtiles
```

Output:

```
Driver: MBTiles/MBTiles
Files: /vsicurl/http://a.tiles.mapbox.com/v3/kkaefer.iceland.mbtiles
Size is 16384, 16384
Coordinate System is:
PROJCS["WGS 84 / Pseudo-Mercator",
    GEOGCS["WGS 84",
        DATUM["WGS_1984",
            SPHEROID["WGS 84",6378137,298.257223563,
                AUTHORITY["EPSG","7030"]],
            AUTHORITY["EPSG","6326"]],
        PRIMEM["Greenwich",0,
            AUTHORITY["EPSG","8901"]],
        UNIT["degree",0.0174532925199433,
            AUTHORITY["EPSG","9122"]],
        AUTHORITY["EPSG","4326"]],
    PROJECTION["Mercator_1SP"],
    PARAMETER["central_meridian",0],
    PARAMETER["scale_factor",1],
    PARAMETER["false_easting",0],
    PARAMETER["false_northing",0],
    UNIT["metre",1,
        AUTHORITY["EPSG","9001"]],
    AXIS["X",EAST],
    AXIS["Y",NORTH],
    EXTENSION["PROJ4","+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_
↪0=0.0 +y_0=0 +k=1.0 +units=m +nadgrids=@null +wktext  +no_defs"],
    AUTHORITY["EPSG","3857"]]
Origin = (-3757031.250000000000000,11271093.750000000000000)
Pixel Size = (152.873992919921875,-152.873992919921875)
Image Structure Metadata:
  INTERLEAVE=PIXEL
Corner Coordinates:
Upper Left  (-3757031.250,11271093.750) ( 33d44'59.95"W, 70d36'45.36"N)
Lower Left  (-3757031.250, 8766406.250) ( 33d44'59.95"W, 61d36'22.97"N)
Upper Right (-1252343.750,11271093.750) ( 11d14'59.98"W, 70d36'45.36"N)
Lower Right (-1252343.750, 8766406.250) ( 11d14'59.98"W, 61d36'22.97"N)
Center      (-2504687.500,10018750.000) ( 22d29'59.97"W, 66d30'47.68"N)
Band 1 Block=256x256 Type=Byte, ColorInterp=Red
  Overviews: 8192x8192, 4096x4096, 2048x2048, 1024x1024, 512x512
  Mask Flags: PER_DATASET ALPHA
  Overviews of mask band: 8192x8192, 4096x4096, 2048x2048, 1024x1024, 512x512
Band 2 Block=256x256 Type=Byte, ColorInterp=Green
  Overviews: 8192x8192, 4096x4096, 2048x2048, 1024x1024, 512x512
  Mask Flags: PER_DATASET ALPHA
  Overviews of mask band: 8192x8192, 4096x4096, 2048x2048, 1024x1024, 512x512
Band 3 Block=256x256 Type=Byte, ColorInterp=Blue
```

(continues on next page)

(continued from previous page)

```
  Overviews: 8192x8192, 4096x4096, 2048x2048, 1024x1024, 512x512
  Mask Flags: PER_DATASET ALPHA
  Overviews of mask band: 8192x8192, 4096x4096, 2048x2048, 1024x1024, 512x512
Band 4 Block=256x256 Type=Byte, ColorInterp=Alpha
  Overviews: 8192x8192, 4096x4096, 2048x2048, 1024x1024, 512x512
```

- Reading pixel attributes encoded according to the UTFGrid specification :

```
$ gdallocationinfo /vsicurl/http://a.tiles.mapbox.com/v3/mapbox.geography-class.
↪mbtiles -wgs84 2 49 -b 1 -xml
```

Output:

```
<Report pixel="33132" line="22506">
  <BandReport band="1">
    <LocationInfo>
      <Key>74</Key>
      <JSon>{"admin":"France","flag_png":
↪"iVBORw0KGgoAAAANSUhEUgAAAGQAAABDEAIAAAC1uevOAAAACXBIWXMAAABIAAAASABGyWs+AAAABmJLR0T/
↪//////
↪8JWPfcAAABPklEQVR42u3cMRLBQBSA4Zc9CgqcALXC4bThBA5gNFyFM+wBVNFqjYTszpfi1Sm++bOv2ETEdNK2pc/
↪T9ny977rCn+fx8rjtc7dMmybnxXy9KncGWGCBBRZYYIEFFlhggQUWWGCBBRZYYIE1/
↪GzSLB0CLLAUCyywwAILLLDAAgsssGyFlcAqnJRiKRZYYIEFFlhggQUWWGDZCsFSLLDAAgsssP4DazQowVIssMACy1ZYG6wP3C
↪HAYWWIplKwQLLLDAAgssZyywwAILLLDAqh6We4VgKZatECywFAsssMACCyywwAILLLBshWCBpVhggQUWWGCBBRZYYIFlKwQLL
↪fp8BhZYigUWWGB9C+t9ggUWWGD5FA44XxBz7mcwZM9VAAAAJXRFWHRkYXRlOmNyZWF0ZQAyMDExLTA5LTAyVDIzOjI5OjIxLT
↪"}</JSon>
    </LocationInfo>
    <Value>238</Value>
  </BandReport>
</Report>
```

- Converting a dataset to MBTiles and adding overviews :

```
$ gdal_translate my_dataset.tif my_dataset.mbtiles -of MBTILES
$ gdaladdo -r average my_dataset.mbtiles 2 4 8 16
```

- Opening a vector MBTiles:

```
$ ogrinfo /home/even/gdal/data/mvt/out.mbtiles
INFO: Open of `/home/even/gdal/data/mvt/out.mbtiles'
      using driver `MBTiles' successful.
Metadata:
  ZOOM_LEVEL=5
  name=out.mbtiles
  description=out.mbtiles
  version=2
  minzoom=0
  maxzoom=5
  center=16.875000,44.951199,5
  bounds=-180.000000,-85.051129,180.000000,83.634101
  type=overlay
  format=pbf
1: ne_10m_admin_1_states_provinces_shpgeojson (Multi Polygon)
```

- Converting a GeoPackage to a Vector tile MBTILES:

```
$ ogr2ogr -f MBTILES target.mbtiles source.gpkg -dsco MAXZOOM=10
```

6.93.12 See Also

- MBTiles specification
- UTFGrid specification
- *Mapbox Vector tiles driver*

6.94 MEM -- In Memory datasets

Driver short name

MEM

Driver built-in by default

This driver is built-in by default

GDAL supports the ability to hold datasets in a temporary in-memory format. This is primarily useful for temporary datasets in scripts or internal to applications. It is not generally of any use to application end-users.

This page documents its raster capabilities. Starting with GDAL 3.11, this driver has been unified with the long-time existing Memory driver to offer both raster and *vector* capabilities.

Memory datasets should support for most kinds of auxiliary information including metadata, coordinate systems, georeferencing, GCPs, color interpretation, nodata, color tables and all pixel data types.

6.94.1 Dataset Name Format

It is possible to open an existing array in memory. To do so, construct a dataset name with the following format:

```
MEM:::option=value[,option=value...]
```

For example:

```
MEM:::DATAPOINTER=342343408,PIXELS=100,LINES=100,BANDS=3,DATATYPE=Byte,
     PIXELOFFSET=3,LINEOFFSET=300,BANDOFFSET=1,
     GEOTRANSFORM=1.166396e+02/1.861068e-05/0.000000e+00/3.627969e+01/0.000000e+00/-1.
↪861068e-05
```

or

```
MEM:::DATAPOINTER=0x1467BEF0,PIXELS=100,LINES=100,BANDS=3,DATATYPE=Byte,
     PIXELOFFSET=3,LINEOFFSET=300,BANDOFFSET=1,
     GEOTRANSFORM=1.166396e+02/1.861068e-05/0.000000e+00/3.627969e+01/0.000000e+00/-1.
↪861068e-05
```

- DATAPOINTER: address of the first pixel of the first band. The address can be represented as a hexadecimal or decimal value. Hexadecimal values must be prefixed with '0x'. Some implementations (notably Windows) doesn't

print hexadecimal pointer values with a leading '0x', so the prefix must be added. You can use CPLPrintPointer to create a string with format suitable for use as a DATAPOINTER.

- PIXELS: Width of raster in pixels. (required)
- LINES: Height of raster in lines. (required)
- BANDS: Number of bands, defaults to 1. (optional)
- DATATYPE: Name of the data type, as returned by GDALGetDataTypeName() (eg. Byte, Int16) Defaults to Byte. (optional)
- PIXELOFFSET: Offset in bytes between the start of one pixel and the next on the same scanline. (optional)
- LINEOFFSET: Offset in bytes between the start of one scanline and the next. (optional)
- BANDOFFSET: Offset in bytes between the start of one bands data and the next.
- GEOTRANSFORM: Set the affine transformation coefficients. 6 real numbers with '/' as separator (optional)
- SPATIALREFERENCE: (GDAL >= 3.7) Set the projection. The coordinate reference systems that can be passed are anything supported by the OGRSpatialReference.SetFromUserInput() as per '-a_srs' in *gdal_translate*. If the passed string includes comma or double-quote characters (typically WKT), it should be surrounded by double-quote characters and the double-quote characters inside it should be escaped with anti-slash. e.g `SPATIALREFERENCE="GEOGCRS[\"WGS 84\",[... snip ...],ID[\"EPSG\",4326]]"`

Warning

Starting with GDAL 3.10, opening a MEM dataset using the above syntax is no longer enabled by default for security reasons. If you want to allow it, define the `GDAL_MEM_ENABLE_OPEN` configuration option to `YES`, or build GDAL with the `GDAL_MEM_ENABLE_OPEN` compilation definition.

GDAL_MEM_ENABLE_OPEN=[YES/NO]: (GDAL >= 3.10) Defaults to `NO`. Whether opening a MEM dataset with the `MEM:::` syntax is allowed.

6.94.2 Creation Options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). This driver supports the following creation option:

- **INTERLEAVE=[BAND/PIXEL]:** Defaults to `BAND`. Set the interleaving to use

The MEM format is one of the few that supports the AddBand() method. The AddBand() method supports DATAPOINTER, PIXELOFFSET and LINEOFFSET options to reference an existing memory array.

6.94.3 Driver capabilities

Supports CreateCopy()

This driver supports the *`GDALDriver::CreateCopy()`* operation

Supports Create()

This driver supports the *`GDALDriver::Create()`* operation

Supports Georeferencing

This driver supports georeferencing

6.94.4 Multidimensional API support

Added in version 3.1.

The MEM driver supports the *Multidimensional Raster Data Model*.

6.95 MFF -- Vexcel MFF Raster

Driver short name

MFF

Driver built-in by default

This driver is built-in by default

GDAL includes read support for Vexcel's MFF raster format. MFF dataset consist of a header file (typically with the extension .hdr) and a set of data files with extensions like .x00, .b00 and so on. To open a dataset select the .hdr file.

Reading lat/long GCPs (TOP_LEFT_CORNER, ...) is supported but there is no support for reading affine georeferencing or projection information.

Unrecognized keywords from the .hdr file are preserved as metadata.

All data types with GDAL equivalents are supported, including 8, 16, 32 and 64 bit data precisions in integer, real and complex data types. In addition tile organized files (as produced by the Vexcel SAR Processor - APP) are supported for reading.

MFF files are not normally portable between systems with different byte orders. However GDAL honours the new BYTE_ORDER keyword which can take a value of LSB (Integer - little endian), and MSB (Motorola - big endian). This may be manually added to the .hdr file if required.

NOTE: Implemented as mffdataset.cpp.

6.95.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.96 MFF2 -- Vexcel MFF2 Image

Driver short name

MFF2

Driver built-in by default

This driver is built-in by default

GDAL supports MFF2 Image raster file format for read. The MFF2 (Multi-File Format 2) format was designed to fit into Vexcel Hierarchical Key-Value (HKV) databases, which can store binary data as well as ASCII parameters. This format is primarily used internally to the Vexcel InSAR processing system.

To select an MFF2 dataset, select the directory containing the `attrib`, and `image_data` files for the dataset.

Currently only latitude/longitude and UTM projection are supported (georef.projection.name = ll or georef.projection.name = utm), with the affine transform computed from the lat/long control points. In any event, if GCPs are available in a georef file, they are returned with the dataset.

For read, all data types (real, integer and complex in bit depths of 8, 16, 32) should be supported.

NOTE: Implemented as hkvdataset.cpp.

6.96.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.96.2 Format Details

6.96.2.1 MFF2 Top-level Structure

An MFF2 "file" is actually a set of files stored in a directory containing an ASCII header file entitled "attrib", and binary image data entitled "image_data". Optionally, there may be an ASCII "georef" file containing georeferencing and projection information, and an "image_data_ovr" (for "image_data" binary image data) file containing tiled overviews of the image in TIFF format. The ASCII files are arranged in key=value pairs. The allowable pairs for each file are described below.

6.96.2.2 The "attrib" File

As a minimum, the "attrib" file must specify the image extents, pixel size in bytes, pixel encoding and datatype, and pixel byte order. For example,

```
extent.cols     = 800
extent.rows     = 1040
```

(continues on next page)

(continued from previous page)

```
pixel.size     = 32
pixel.encoding = { unsigned twos_complement *ieee_754 }
pixel.field    = { *real complex }
pixel.order    = { lsbf *msbf }
version        = 1.1
```

specifies an image that is 1040 lines by 800 pixels in extent. The pixels are 32 bits of real data in "most significant byte first" (msbf) order, encoded according to the ieee_754 specification. In MFF2, when a value must belong to a certain subset (eg. pixel.order must be either lsbf or msbf), all options are displayed between curly brackets, and the one appropriate for the current file is indicated with a "*".

The file may also contain the following lines indicating the number of channels of data, and how they are interleaved within the binary data file.

```
channel.enumeration = 1
channel.interleave = { *pixel tile sequential }
```

6.96.2.3 The "image_data" File

The "image_data" file consists of raw binary data, with extents, pixel encoding, and number of channels as indicated in the "attrib" file.

6.96.2.4 The "georef" File

The "georef" file is used to describe the geocoding and projection information for the binary data. For example,

```
top_left.latitude            = 32.93333333333334
top_left.longitude           = 130.0
top_right.latitude           = 32.93333333333334
top_right.longitude          = 130.5
bottom_left.latitude         = 32.50000000000001
bottom_left.longitude        = 130.0
bottom_right.latitude        = 32.50000000000001
bottom_right.longitude       = 130.5
centre.latitude              = 32.71666666666668
centre.longitude             = 130.25
projection.origin_longitude  = 0
projection.name              = ll
spheroid.name                = wgs-84
```

describes an orthogonal latitude/longitude (ll) projected image, with latitudes and longitudes based on the wgs-84 ellipsoid.

Since MFF2 version 1.1, top_left refers to the top left corner of the top left pixel. top_right refers to the top right corner of the top right pixel. bottom_left refers to the bottom left corner of the bottom left pixel. bottom_right refers to the bottom right corner of the bottom right pixel. centre refers to the centre of the four corners defined above (center of the image).

Mathematically, for an Npix by Nline image, the corners and centre in (pixel,line) coordinates for MFF2 version 1.1 are:

```
top_left: (0,0)
top_right: (Npix,0)
bottom_left: (0,Nline)
```

(continues on next page)

(continued from previous page)

```
bottom_right: (Npix,Nline)
centre: (Npix/2.0,Nline/2.0)
```

These calculations are done using floating point arithmetic (i.e. centre coordinates may take on non-integer values).

Note that the corners are always expressed in latitudes/longitudes, even for projected images.

6.96.2.5 Supported projections

ll- Orthogonal latitude/longitude projected image, with latitude parallel to the rows, longitude parallel to the columns. Parameters: spheroid name, projection.origin_longitude (longitude at the origin of the projection coordinates). If not set, this should default to the central longitude of the output image based on its projection boundaries.

utm- Universal Transverse Mercator projected image. Parameters: spheroid name, projection.origin_longitude (central meridian for the utm projection). The central meridian must be the meridian at the centre of a UTM zone, i.e. 3 degrees, 9 degrees, 12 degrees, etc. If this is not specified or set a valid UTM central meridian, the reader should reset the value to the nearest valid central meridian based on the central longitude of the output image. The latitude at the origin of the UTM projection is always 0 degrees.

6.96.2.6 Recognized ellipsoids

MFF2 format associates the following names with ellipsoid equatorial radius and inverse flattening parameters:

```
airy-18304:             6377563.396      299.3249646
modified-airy4:         6377340.189      299.3249646
australian-national4:   6378160          298.25
bessel-1841-namibia4:   6377483.865      299.1528128
bessel-18414:           6377397.155      299.1528128
clarke-18584:           6378294.0        294.297
clarke-18664:           6378206.4        294.9786982
clarke-18804:           6378249.145      293.465
everest-india-18304:    6377276.345      300.8017
everest-sabah-sarawak4:6377298.556       300.8017
everest-india-19564:    6377301.243      300.8017
everest-malaysia-19694:6377295.664       300.8017
everest-malay-sing4:    6377304.063      300.8017
everest-pakistan4:      6377309.613      300.8017
modified-fisher-19604: 6378155           298.3
helmert-19064:          6378200          298.3
hough-19604:            6378270          297
hughes4:                6378273.0        298.279
indonesian-1974:        6378160          298.247
international-1924:     6378388          297
iugc-67:                6378160.0        298.254
iugc-75:                6378140.0        298.25298
krassovsky-1940:        6378245          298.3
kaula:                  6378165.0        292.308
grs-80:                 6378137          298.257222101
south-american-1969:    6378160          298.25
wgs-72:                 6378135          298.26
wgs-84:                 6378137          298.257223563
ev-wgs-84:              6378137          298.252841
ev-bessel:              6377397          299.1976073
```

6.96.2.7 Explanation of fields

```
channel.enumeration:  (optional- only needed for multiband)
Number of channels of data (eg. 3 for rgb)

channel.interleave = { *pixel tile sequential } :  (optional- only
needed for multiband)

For multiband data, indicates how the channels are interleaved.  *pixel
indicates that data is stored red value, green value, blue value, red
value, green value, blue value etc. as opposed to (line of red values)
(line of green values) (line of blue values) or (entire red channel)
(entire green channel) (entire blue channel)

extent.cols:
Number of columns of data.

extent.rows:
Number of rows of data.

pixel.encoding = { *unsigned twos-complement ieee-754 }:
Combines with pixel.size and pixel.field to give the data type:
(encoding, field, size)- type
(unsigned, real, 8)- unsigned byte data
(unsigned, real, 16)- unsigned int 16 data
(unsigned, real, 32)- unsigned int 32 data
(twos-complement, real, 16)- signed int 16 data
(twos-complement, real, 32)- signed int 32 data
(twos-complement, complex, 64)- complex signed int 32 data
(ieee-754, real, 32)- real 32 bit floating point data
(ieee-754, real, 64)- real 64 bit floating point data
(ieee-754, complex, 64)- complex 32 bit floating point data
(ieee-754, complex, 128)- complex 64 bit floating point data

pixel.size:
Size of one pixel of one channel (bits).

pixel.field = { *real complex }:
Whether the data is real or complex.

pixel.order = { *lsbf msbf }:
Byte ordering of the data (least or most significant byte first).

version: (only in newer versions- if not present, older version is
assumed) Version of mff2.
```

6.97 MiraMon Raster

Added in version 3.12.

Driver short name

MiraMonRaster

Driver built-in by default

This driver is built-in by default

This driver is capable of reading raster files in MiraMon format as well as create a copy to MiraMon format.

A look-up table of MiraMon and EPSG Spatial Reference Systems allows matching identifiers in both systems.

6.97.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.97.2 Overview of MiraMon format

The MiraMon *.img* format is a binary raster format with rich metadata stored in a sidecar *I.rel* file. More information is available in the public specification.

By specifying either the name of the *I.rel* metadata file or the name of any *.img* band file, the driver will automatically use the associated *I.rel* file.

- **REL file**: This metadata file governs how all bands are interpreted and accessed. It contains metadata including band names, number of rows and columns, data type, compression information (either global or per band) and others. So, a MiraMon dataset can include multiple bands, all linked through a single *.rel* file. Whether the ".img" file or the "I.rel" metadata file is provided, the result will be the same: all bands will be considered. If a layer contains an old *.rel* format file (used in legacy datasets), a warning will be issued explaining how to convert it into the modern *.rel 4* format. Next are the main characteristics of a MiraMon raster dataset band:
- **IMG file**: Stores the raw raster data. The data type may vary: - *Bit*: 1 bit. Range: 0 or 1. Converted to byte GDAL type. - *Byte*: 1 byte, unsigned. Range: 0 to 255 - *Short*: 2 bytes, signed. Range: -32 768 to 32 767 - *UShort*: 2 bytes, unsigned. Range: 0 to 65 535 - *Int32*: 4 bytes, signed. Range: -2 147 483 648 to 2 147 483 647 - *Real*: 4 bytes, floating-point - *Double*: 8 bytes, double precision floating-point

It's important that the specified filename for the dataset is "I.rel" instead of ".rel". Files ending with ".img" can be confused with the raw data files, and the chosen driver may not be the correct one. If the filename is "I.rel", the driver will automatically be MiraMonRaster. If ".img" is used, the user can also specify the driver with the open option "-if MiraMonRaster" or "-of MiraMonRaster".

6.97.3 Writing behavior

If the dataset has documented R, G, and B bands, the copy will also generate a map (in .mmm format) to allow the file to be visualized in MiraMon as an RGB raster (24-bit).

6.97.4 Color Table

MiraMon raster bands may include a color table in dBASE (DBF) format, or in extended DBF format, if necessary. If present, it will be exposed through the `GetRasterColorTable()` method. More information is available in the public specification.

6.97.5 Attribute Table

MiraMon raster bands may also include an attribute table in the same format as color table. If available, it will be exposed as a GDAL Raster Attribute Table (RAT) via `GetDefaultRAT()`. This allows applications to retrieve additional metadata associated with each raster value, such as class names or descriptions.

6.97.6 Metadata

Metadata from MiraMon raster datasets are exposed through the "MIRAMON" domain. Only a subset of the metadata is used by the driver to interpret the dataset (e.g., georeferencing, data type, etc), but all other metadata entries are preserved and accessible via `GetMetadata("MIRAMON")`.

This allows applications to access additional information embedded in the original dataset, even if it is not required for reading or displaying the data (use of -co SRC_MDD=MIRAMON will copy to destination dataset all this MIRAMON metadata).

When creating a copy, some MIRAMON metadata items are copied to the comments part of REL file. Also, when creating a copy, lineage metadata (if existent) is recovered from the source dataset and added to the MiraMon file to know how the file was created. The current process is added after all recovered processes to document that the file was created by GDAL.

6.97.7 Encoding

When reading MiraMon DBF files, the code page setting in the *.dbf* header is used to translate string fields to UTF-8, regardless of whether the original encoding is ANSI, OEM or UTF-8.

REL files are always encoded in ANSI.

6.97.8 Subdataset Generation

The MiraMon format allows datasets to contain bands with heterogeneous characteristics. At present, the only strict restriction imposed by the format itself is that all bands must share the same spatial reference system (although different spatial extents and/or cell sizes are allowed).

For interoperability with other GDAL formats, the MiraMonRaster driver applies, by default, a compatibility-based criterion when exposing multiband datasets. Bands are grouped into the same GDAL subdataset only when they are fully compatible. When this is not the case, bands are exposed as belonging to different subdatasets.

In practice, bands are separated into different subdatasets when there are differences in raster geometry, including raster dimensions or spatial extent, when the numeric data type differs, when symbolization semantics differ, when associated Raster Attribute Tables differ, or when the presence or value of NoData differs.

6.97.9 Open options

The following open options are available:

- **RAT_OR_CT=[ALL/RAT/CT]:** Defaults to `ALL`. Controls whether the Raster Attribute Table (RAT) and/or the Color Table (CT) are exposed.

  ALL
  Expose both the attribute table and the color table. Note that in some software this option may cause visualization and/or legend issues.

  RAT
  Expose the attribute table only, without the color table.

  CT
  Expose the color table only, without the attribute table.

6.97.10 Creation Options

The following creation options are supported:

- **COMPRESS=[YES/NO]:** Defaults to `YES`. Indicates whether the file will be compressed in RLE indexed mode.
- **PATTERN=value:** Indicates the pattern used to create the names of the different bands. In the case of RGB, the suffixes "_R", "_G", and "_B" will be added to the base name.
- **CATEGORICAL_BANDS=<integer>:** Indicates which bands (1-based indexing) have to be treated as categorical (e.g., a land cover map). Comma (",") separators can be used to indicate multiple bands.
- **CONTINUOUS_BANDS=<integer>:** Indicates which bands (1-based indexing) have to be treated as continuous (e.g., temperatures map). Comma (",") separators can be used to indicate multiple bands. If a band is not indicated as categorical or continuous, it will be treated following an automatic criterion based on the presence of a color table and/or an attribute table, for instance.

6.97.11 Open examples

- A MiraMon dataset with 3 bands that have different spatial extents and/or cell sizes will be exposed as 3 different subdatasets. This allows applications to read each band independently, without the need to resample them to a common grid. You can use `-sds` option to translate all datasets into, for instance, different TIFF files: .. code-block:

  ```
  gdal_translate multiband_input_datasetI.rel output_subdatasets.tiff -sds
  ```

  Output: output_subdatasets_1.tiff, output_subdatasets_2.tiff, output_subdatasets_3.tiff

- A MiraMon dataset color table and RAT can be translated separately, by using the RAT_OR_CT open option: .. code-block:

  ```
  gdal_translate -oo RAT_OR_CT=RAT datasetI.rel output_only_with_rat.tiff (only RAT␣
  ↪will be translated, without the color table)
  gdal_translate -oo RAT_OR_CT=CT datasetI.rel output_only_with_ct.tiff (only color␣
  ↪table will be translated, without the attribute table)
  ```

- A MiraMon dataset can be translated preserving it's own metadata, by using the SRC_MDD creation option: .. code-block:

  ```
  gdal_translate -oo SRC_MDD=MIRAMON datasetI.rel output_with_miramon_metadata.vrt
  ```

6.97.12 Creation examples

- A tiff file will be translated as a compressed file. If user does not want this behavior COMPRESS=NO can be specified in the creation options: .. code-block:

```
gdal_translate -co COMPRESS=NO dataset.tiff output_uncompressed_datasetI.rel
```

```
gdal_translate -co PATTERN="band_" input.tiff outputI.rel
```

  Output: band_1I.rel, band_2I.rel, band_3I.rel

- A tiff dataset with 3 bands that wants all bands to be treated as categorical will be translated as follows: .. code-block:

```
gdal_translate -co Categorical=1,2,3 dataset.tiff outputI.rel
```

  Output: output_1I.rel, output_2I.rel, output_3I.rel

- A tiff dataset with 3 bands that wants the first band to be treated as categorical will be translated as follows: .. code-block:

```
gdal_translate -co Categorical=1 dataset.tiff outputI.rel
```

  Output: output_1I.rel, output_2I.rel, output_3I.rel Bands 2 and 3 will be treated with a heuristic criterion, for instance, based on the presence of a color table and/or an attribute table.

- A tiff dataset with 3 bands that wants the first band to be treated as categorical and the second one as continuous will be translated as follows: .. code-block:

```
gdal_translate -co Categorical=1 -co Continuous=2 dataset.tiff outputI.rel
```

  Output: output_1I.rel, output_2I.rel, output_3I.rel Band 1 will be treated as categorical, band 2 as continuous, and band 3 with a heuristic criterion, for instance, based on the presence of a color table and/or an attribute table.

- A MiraMon dataset can be translated from a source and retrieving all its metadata, by using the SRC_MDD creation option: .. code-block:

```
gdal_translate -oo SRC_MDD=MIRAMON input_with_miramon_metadata.vrt MiraMonDatasetI.
↪rel
```

6.97.13 See Also

- MiraMon's raster format specifications
- MiraMon Extended DBF format
- MiraMon raster layer concepts.
- MiraMon main page
- MiraMon help guide
- Grumets research group, the people behind MiraMon

6.98 MrSID -- Multi-resolution Seamless Image Database

Driver short name

MrSID

Build dependencies

MrSID SDK

MrSID is a wavelet-based image compression technology which can utilize both lossy and lossless encoding. This technology was acquired in its original form from Los Alamos National Laboratories (LANL), where it was developed under the aegis of the U.S. government for storing fingerprints for the FBI. Now it is developed and distributed by Extensis.

This driver supports reading of MrSID image files using Extensis' decoding software development kit (DSDK). **This DSDK is not free software, you should contact Extensis to obtain it (see link at end of this page).** If you are using GCC, please, ensure that you have the same compiler as was used for DSDK compilation. It is C++ library, so you may get incompatibilities in C++ name mangling between different GCC versions (2.95.x and 3.x).

Latest versions of the DSDK also support decoding JPEG2000 file format, so this driver can be used for JPEG2000 too.

6.98.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.98.2 Metadata

MrSID metadata transparently translated into GDAL metadata strings. Files in MrSID format contain a set of standard metadata tags such as: IMAGE__WIDTH (contains the width of the image), IMAGE__HEIGHT (contains the height of the image), IMAGE__XY_ORIGIN (contains the x and y coordinates of the origin), IMAGE__INPUT_NAME (contains the name or names of the files used to create the MrSID image) etc. GDAL's metadata keys cannot contain characters `:' and `=', but standard MrSID tags always contain double colons in tag names. These characters replaced in GDAL with `_' during translation. So if you are using other software to work with MrSID be ready that names of metadata keys will be shown differently in GDAL.

XMP metadata can be extracted from JPEG2000 files, and will be stored as XML raw content in the xml:XMP metadata domain.

6.98.3 Georeference

MrSID images may contain georeference and coordinate system information in form of GeoTIFF GeoKeys, translated in metadata records. All those GeoKeys properly extracted and used by the driver. Unfortunately, there is one caveat: old MrSID encoders has a bug which resulted in wrong GeoKeys, stored in MrSID files. This bug was fixed in MrSID software version 1.5, but if you have older encoders or files, created with older encoders, you cannot use georeference information from them.

6.98.4 Standalone plugin compilation

Added in version 3.10.

While this driver may be built as part of a whole GDAL build, either in libgdal itself, or as a plugin, it is also possible to only build this driver as a plugin, against an already built libgdal.

The version of the GDAL sources used to build the driver must match the version of the libgdal it is built against.

For example, from a "build_mrsid" directory under the root of the GDAL source tree:

```
cmake -S ../frmts/mrsid -DCMAKE_PREFIX_PATH=/path/to/GDAL_installation_prefix -DMRSID_
↪ROOT=/path/to/mrsid_sdk_root
cmake --build .
```

Note that such a plugin, when used against a libgdal not aware of it, will be systematically loaded at GDAL driver initialization time, and will not benefit from *deferred plugin loading capabilities <rfc-96>*. For that, libgdal itself must be built with the CMake variable GDAL_REGISTER_DRIVER_MRSID_FOR_LATER_PLUGIN=ON set.

6.98.5 See Also:

- Implemented as mrsiddataset.cpp.
- Extensis web site

6.99 MSG -- Meteosat Second Generation

Driver short name

MSG

Build dependencies

msg library

This driver implements reading support for Meteosat Second Generation files. These are files with names like `H-000-MSG1\_\_-MSG1\_\_\_\_\_\_\_\_-HRV\_\_\_\_\_-000007\_\_\_-200405311115-C\_`, commonly distributed into a folder structure with dates (e.g. `2004\05\31` for the file mentioned).

The MSG files are wavelet-compressed. A decompression library licensed from EUMETSAT is needed (Public Wavelet Transform Decompression Library Software, shorter *Wavelet Transform Software*). The software is compilable on Microsoft Windows, Linux and Solaris Operating Systems, and it works on 32 bits and 64 bits as well as mixed architectures. It is licensed under Apache v2.

This driver is not "enabled" by default. See *Build Instructions* on how to include this driver in your GDAL library.

6.99.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

6.99.2 Build Instructions

See the `GDAL_USE_PUBLICDECOMPWT` option of building_from_source.

6.99.3 Specification of Source Dataset

It is possible to select individual files for opening. In this case, the driver will gather the files that correspond to the other strips of the same image, and correctly compose the image.

Example with gdal_translate.exe:

```
gdal_translate
 C:\hrit_a\2004\05\31\H-000-MSG1__-MSG1________-HRV______-000008___-200405311115-C_
 C:\output\myimage.tif
```

It is also possible to use the following syntax for opening the MSG files:

- MSG(source_folder,timestamp,(channel,channel,...,channel),use_root_folder,data_conversion,nr_cycles,step)
-
  - source_folder: a path to a folder structure that contains the files
  - timestamp: 12 digits representing a date/time that identifies the 114 files of the 12 images of that time, e.g. 200501181200
  - channel: a number between 1 and 12, representing each of the 12 available channels. When only specifying one channel, the brackets are optional.
  - use_root_folder: Y to indicate that the files reside directly into the source_folder specified. N to indicate that the files reside in date structured folders: source_folder/YYYY/MM/DD
  - data_conversion:
  -
    - N to keep the original 10 bits DN values. The result is UInt16.
    - B to convert to 8 bits (handy for GIF and JPEG images). The result is Byte.
    - R to perform radiometric calibration and get the result in mW/m2/sr/(cm-1)-1. The result is Float32.
    - L to perform radiometric calibration and get the result in W/m2/sr/um. The result is Float32.
    - T to get the reflectance for the visible bands (1, 2, 3 and 12) and the temperature in degrees Kelvin for the infrared bands (all other bands). The result is Float32.
  - nr_cycles: a number that indicates the number of consecutive cycles to be included in the same file (time series). These are appended as additional bands.
  - step: a number that indicates what is the stepsize when multiple cycles are chosen. E.g. every 15 minutes: step = 1, every 30 minutes: step = 2 etc. Note that the cycles are exactly 15 minutes apart, so you can not get images from times in-between (the step is an integer).

Examples with gdal_translate utility:

Example call to fetch an MSG image of 200501181200 with bands 1, 2 and 3 in IMG format:

```
gdal_translate -of HFA MSG(\\pc2133-24002\RawData\,200501181200,(1,2,3),N,N,1,1) d:\
↪output\outfile.img
```

In JPG format, and converting the 10 bits image to 8 bits by dividing all values by 4:

```
gdal_translate -of JPEG MSG(\\pc2133-24002\RawData\,200501181200,(1,2,3),N,B,1,1) d:\
↪output\outfile.jpg
```

The same, but reordering the bands in the JPEG image to resemble RGB:

```
gdal_translate -of JPEG MSG(\\pc2133-24002\RawData\,200501181200,(3,2,1),N,B,1,1) d:\
↪output\outfile.jpg
```

Geotiff output, only band 2, original 10 bits values:

```
gdal_translate -of GTiff MSG(\\pc2133-24002\RawData\,200501181200,2,N,N,1,1) d:\output\
↪outfile.tif
```

Band 12:

```
gdal_translate -of GTiff MSG(\\pc2133-24002\RawData\,200501181200,12,N,N,1,1) d:\output\
↪outfile.tif
```

The same band 12 with radiometric calibration in mW/m2/sr/(cm-1)-1:

```
gdal_translate -of GTiff MSG(\\pc2133-24002\RawData\,200501181200,12,N,R,1,1) d:\output\
↪outfile.tif
```

Retrieve data from c:hrit-data20050118 instead of \pc2133-24002RawData... :

```
gdal_translate -of GTiff MSG(c:\hrit-data\2005\01\18,200501181200,12,Y,R,1,1) d:\output\
↪outfile.tif
```

Another option to do the same (note the difference in the Y and the N for the "use_root_folder" parameter:

```
gdal_translate -of GTiff MSG(c:\hrit-data\,200501181200,12,N,R,1,1) d:\output\outfile.tif
```

Without radiometric calibration, but for 10 consecutive cycles (thus from 1200 to 1415):

```
gdal_translate -of GTiff MSG(c:\hrit-data\,200501181200,12,N,N,10,1) d:\output\outfile.
↪tif
```

10 cycles, but every hour (thus from 1200 to 2100):

```
gdal_translate -of GTiff MSG(c:\hrit-data\,200501181200,12,N,N,10,4) d:\output\outfile.
↪tif
```

10 cycles, every hour, and bands 3, 2 and 1:

```
gdal_translate -of GTiff MSG(c:\hrit-data\,200501181200,(3,2,1),N,N,10,4) d:\output\
↪outfile.tif
```

6.99.4 Georeference and Projection

The images are using the Geostationary Satellite View projection. Most GIS packages don't recognize this projection (we only know of ILWIS that does have this projection), but gdalwarp.exe can be used to re-project the images.

6.99.5 See Also

- Implemented as msgdataset.cpp.
- http://www.eumetsat.int - European Organisation for the Exploitation of Meteorological Satellites

6.100 MSGN -- Meteosat Second Generation (MSG) Native Archive Format (.nat)

Driver short name

MSGN

Driver built-in by default

This driver is built-in by default

GDAL supports reading only of MSG native files. These files may have anything from 1 to 12 bands, all at 10-bit resolution.

Includes support for the 12th band (HRV - High Resolution Visible). This is implemented as a subset, i.e., it is accessed by prefixing the filename with the tag "HRV:".

Similarly, it is possible to obtain floating point radiance values in stead of the usual 10-bit digital numbers (DNs). This subset is accessed by prefixing the filename with the tag "RAD:".

Georeferencing is currently supported, but the results may not be acceptable (accurate enough), depending on your requirements. The current workaround is to implement the CGMS Geostationary projection directly, using the code available from EUMETSAT.

6.100.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.101 NDF -- NLAPS Data Format

Driver short name

NDF

Driver built-in by default

This driver is built-in by default

GDAL has limited support for reading NLAPS Data Format files. This is a format primarily used by the Eros Data Center for distribution of Landsat data. NDF datasets consist of a header file (often with the extension .H1) and one or more associated raw data files (often .I1, .I2, ...). To open a dataset select the header file, often with the extension .H1, .H2 or .HD.

The NDF driver only supports 8bit data. The only supported projection is UTM. NDF version 1 (NDF_VERSION=0.00) and NDF version 2 are both supported.

NOTE: Implemented as ndfdataset.cpp.

See Also: NLAPS Data Format Specification.

6.101.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.102 NetCDF: Network Common Data Form

Driver short name

netCDF

Build dependencies

libnetcdf

This format is supported for read and write access. This page only describes the raster support (you can find documentation for the *vector side*) NetCDF is an interface for array-oriented data access and is used for representing scientific data.

The fill value metadata or missing_value backward compatibility is preserved as NODATA value when available.

NOTE: Implemented as netcdfdataset.cpp.

6.102.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

6.102.2 Multiple Image Handling (Subdatasets)

Network Common Data Form is a container for several different arrays most used for storing scientific dataset. One NetCDF file may contain several datasets. They may differ in size, number of dimensions and may represent data for different regions.

If the file contains only one NetCDF array which appears to be an image, it may be accessed directly, but if the file contains multiple images it may be necessary to import the file via a two step process.

The first step is to get a report of the components images (dataset) in the file using gdalinfo, and then to import the desired images using gdal_translate. The gdalinfo utility lists all multidimensional subdatasets from the input NetCDF file.

The name of individual images are assigned to the SUBDATASET_n_NAME metadata item. The description for each image is found in the SUBDATASET_n_DESC metadata item. For NetCDF images will follow this format: *NETCDF:filename:variable_name*

where *filename* is the name of the input file, and *variable_name* is the dataset selected within the file.

On the second step you provide this name for **gdalinfo** to get information about the dataset or **gdal_translate** to read dataset.

For example, we want to read data from a NetCDF file:

```
$ gdalinfo sst.nc
Driver: netCDF/Network Common Data Format
Size is 512, 512
Coordinate System is `'
Metadata:
  NC_GLOBAL#title=IPSL  model output prepared for IPCC Fourth Assessment SRES A2␣
↪experiment
  NC_GLOBAL#institution=IPSL (Institut Pierre Simon Laplace, Paris, France)
  NC_GLOBAL#source=IPSL-CM4_v1 (2003) : atmosphere : LMDZ (IPSL-CM4_IPCC, 96x71x19) ;␣
↪ocean ORCA2 (ipsl_cm4_v1_8, 2x2L31); sea ice LIM (ipsl_cm4_v
  NC_GLOBAL#contact=Sebastien Denvil, sebastien.denvil@ipsl.jussieu.fr
  NC_GLOBAL#project_id=IPCC Fourth Assessment
```

(continues on next page)

(continued from previous page)

```
  NC_GLOBAL#table_id=Table O1 (13 November 2004)
  NC_GLOBAL#experiment_id=SRES A2 experiment
  NC_GLOBAL#realization=1
  NC_GLOBAL#cmor_version=9.600000e-01
  NC_GLOBAL#Conventions=CF-1.0
  NC_GLOBAL#history=YYYY/MM/JJ: data generated; YYYY/MM/JJ+1 data transformed  At␣
↪16:37:23 on 01/11/2005, CMOR rewrote data to comply with CF standards and IPCC Fourth␣
↪Assessment requirements
  NC_GLOBAL#references=Dufresne et al, Journal of Climate, 2015, vol XX, p 136
  NC_GLOBAL#comment=Test drive
Subdatasets:
  SUBDATASET_1_NAME=NETCDF:"sst.nc":lon_bnds
  SUBDATASET_1_DESC=[180x2] lon_bnds (64-bit floating-point)
  SUBDATASET_2_NAME=NETCDF:"sst.nc":lat_bnds
  SUBDATASET_2_DESC=[170x2] lat_bnds (64-bit floating-point)
  SUBDATASET_3_NAME=NETCDF:"sst.nc":time_bnds
  SUBDATASET_3_DESC=[24x2] time_bnds (64-bit floating-point)
  SUBDATASET_4_NAME=NETCDF:"sst.nc":tos
  SUBDATASET_4_DESC=[24x170x180] sea_surface_temperature (32-bit floating-point)Corner␣
↪Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0,  512.0)
Upper Right (  512.0,    0.0)
Lower Right (  512.0,  512.0)
Center      (  256.0,  256.0)
```

This NetCDF files contain 4 datasets, lon_bnds, lat_bnds, tim_bnds and tos. Now select the subdataset, described as: `NETCDF:"sst.nc":tos [24x17x180] sea_surface_temperature (32-bit floating-point)` and get the information about the number of bands there is inside this variable.

```
$ gdalinfo NETCDF:"sst.nc":tos
Driver: netCDF/Network Common Data Format
Size is 180, 170
Coordinate System is `'
Origin = (1.000000,-79.500000)
Pixel Size = (1.98888889,0.99411765)
Metadata:
  NC_GLOBAL#title=IPSL  model output prepared for IPCC Fourth Assessment SRES A2␣
↪experiment
  NC_GLOBAL#institution=IPSL (Institut Pierre Simon Laplace, Paris, France)
```

.... More metadata

```
  time#standard_name=time
  time#long_name=time
  time#units=days since 2001-1-1
  time#axis=T
  time#calendar=360_day
  time#bounds=time_bnds
  time#original_units=seconds since 2001-1-1
Corner Coordinates:
Upper Left  (   1.0000000, -79.5000000)
```

(continues on next page)

(continued from previous page)

```
Lower Left  (   1.0000000,  89.5000000)
Upper Right (     359.000,     -79.500)
Lower Right (     359.000,      89.500)
Center      ( 180.0000000,   5.0000000)
Band 1 Block=180x1 Type=Float32, ColorInterp=Undefined
  NoData Value=1e+20
  Metadata:
    NETCDF_VARNAME=tos
    NETCDF_DIMENSION_time=15
    NETCDF_time_units=days since 2001-1-1
Band 2 Block=180x1 Type=Float32, ColorInterp=Undefined
  NoData Value=1e+20
  Metadata:
    NETCDF_VARNAME=tos
    NETCDF_DIMENSION_time=45
    NETCDF_time_units=days since 2001-1-1
```

.... More Bands

```
Band 22 Block=180x1 Type=Float32, ColorInterp=Undefined
  NoData Value=1e+20
  Metadata:
    NETCDF_VARNAME=tos
    NETCDF_DIMENSION_time=645
    NETCDF_time_units=days since 2001-1-1
Band 23 Block=180x1 Type=Float32, ColorInterp=Undefined
  NoData Value=1e+20
  Metadata:
    NETCDF_VARNAME=tos
    NETCDF_DIMENSION_time=675
    NETCDF_time_units=days since 2001-1-1
Band 24 Block=180x1 Type=Float32, ColorInterp=Undefined
  NoData Value=1e+20
  Metadata:
    NETCDF_VARNAME=tos
    NETCDF_DIMENSION_time=705
    NETCDF_time_units=days since 2001-1-1
```

gdalinfo displays the number of bands into this subdataset. There are metadata attached to each band. In this example, the metadata informs us that each band correspond to an array of monthly sea surface temperature from January 2001. There are 24 months of data in this subdataset. You may also use **gdal_translate** for reading the subdataset.

Note that you should provide exactly the contents of the line marked **SUBDATASET_n_NAME** to GDAL, including the **NETCDF:** prefix.

The **NETCDF** prefix must be first. It triggers the subdataset NetCDF driver. This driver is intended only for importing remote sensing and geospatial datasets in form of raster images. If you want explore all data contained in NetCDF file you should use another tools.

Starting with GDAL 3.5, the *VARIABLES_AS_BANDS=YES* open option can be used to indicate to the driver that if the netCDF file only contains 2D variables of the same type and indexed by the same dimensions, then they should be reported as multiple bands of a same dataset.

```
$ gdalinfo autotest/gdrivers/data/netcdf/two_vars_scale_offset.nc -oo VARIABLES_AS_
↪BANDS=YES

Driver: netCDF/Network Common Data Format
Files: autotest/gdrivers/data/netcdf/two_vars_scale_offset.nc
Size is 21, 21
Metadata:
  NC_GLOBAL#Conventions=COARDS/CF-1.0
  x#actual_range={-10,10}
  x#long_name=x
  y#actual_range={-10,10}
  y#long_name=y
  z#add_offset=1.5
  z#long_name=z
  z#scale_factor=0.01
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0,   21.0)
Upper Right (   21.0,    0.0)
Lower Right (   21.0,   21.0)
Center      (   10.5,   10.5)
Band 1 Block=21x1 Type=Float32, ColorInterp=Undefined
  NoData Value=9.96920996838686905e+36
  Offset: 1.5,   Scale:0.01
  Metadata:
    add_offset=1.5
    long_name=z
    NETCDF_VARNAME=z
    scale_factor=0.01
Band 2 Block=21x1 Type=Float32, ColorInterp=Undefined
  NoData Value=9.96920996838686905e+36
  Offset: 2.5,   Scale:0.1
  Metadata:
    add_offset=2.5
    long_name=q
    NETCDF_VARNAME=q
    scale_factor=0.1
```

Starting with GDAL 3.10, specifying the `-if netCDF` option to command line utilities accepting it, or `netCDF` as the only value of the `papszAllowedDrivers` of *GDALOpenEx()*, also forces the driver to recognize the passed filename, when it is not using subdataset syntax (it can typically be used to force open a HDF5 file that would be nominally recognized by the HDF5 driver).

6.102.3 Dimension Ordering

The NetCDF driver assume that dimensions are ordered as (Z/T,Y,X), according to the recommendations of the CF Conventions. If there are more than 3 dimensions, the driver will merge them into bands. For example, if you have a 4-dimensional array with dimensions (P, T, Y, X), the driver will combine the last 2 dimensions (P*T) into bands. Bands will be ordered by first incrementing T and then P. Metadata will be displayed on each band with its corresponding T and P values.

6.102.4 Georeferencing

6.102.4.1 Spatial reference system

There are multiple ways of storing spatial reference systems in netCDF files. Under the CF conventions, each data variable may use a `grid_mapping` attribute to reference the name of a separate "grid mapping" variable, such as `crs`, whose attributes in turn specify parameters of the spatial reference system. Because the parameters recognized by the CF conventions (`semi_major_axis`, `inverse_flattening`, `standard_parallel`, etc.) may not be adequate to fully describe a spatial reference system, the grid mapping variable may also have a `crs_wkt` attribute that fully describes the spatial reference system using an OGC WKT string. The attribute name `crs_wkt` was not specified until version 1.7 of the CF conventions; before that time, GDAL used the attribute name `spatial_ref` for the same purpose.

When reading a variable from a netCDF file, GDAL first checks the `grid_mapping` attribute to find the name of the grid mapping variable. If this variable has a `spatial_ref` or `crs_wkt` attribute (checked in that order), it will be used to assign a spatial reference system. If not, the spatial reference system will be assigned using using projection parameter attributes (`semi_major_axis`, etc.) If these parameters are not present, GDAL will check the grid mapping variable for a (non-standard) `srid` attribute and attempt to interpret it if present.

If no `grid_mapping` attribute is present, GDAL will check the data variable for a (non-standard) `crs` attribute and read a WKT, EPSG, or PROJ.4 string from it.

If GDAL cannot determine a spatial reference system using the above methods, it will be left undefined unless the dataset was opened with the *`ASSUME_LONGLAT`* option.

6.102.4.2 Geotransform

NetCDF files store locations for each pixel using coordinate variables such as `latitude` and `longitude`. This differs from the GDAL *Raster Data Model*, wherein pixel locations are defined relative to an origin point, with fixed resolutions along two axes. The GDAL model uses a six-parameter geotransform to store the origin point, resolutions, and axis rotations (*Geotransform Tutorial*).

If the netCDF file represents a rectangular grid with equal cell sizes, GDAL can derive a geotransform for the dataset by reading the values of the coordinate variables and computing spacings between adjacent pixels. Because of floating-point errors, this geotransform may not match the resolution of the original dataset. To allow the geotransform to be stored exactly, GDAL writes it as an attribute of the grid mapping variable described above. However, this is a GDAL-specific extension and will not typically be present in files not written by GDAL.

If the grid is rectangular but the cell sizes are not equal, as in the case of Gaussian grids used for weather and climate modelling, GDAL will still report a geotransform if the cell sizes are equal within a 0.1-degree threshold. Whether or not a geotransform is reported, the exact values of the cell centers can be accessed via the `Y_VALUES` metadata item in the `GEOLOCATION2` domain.

If the file does not represent a rectangular grid, GDAL will not report a geotransform. In this case, the `GEOLOCATION` metadata domain provides references to the datasets that can be used to read the coordinates associated with each pixel.

When reading a netCDF, GDAL must first identify the coordinate variables associated with a data variable. The configuration options *`GDAL_NETCDF_VERIFY_DIMS`* and *`GDAL_NETCDF_IGNORE_XY_AXIS_NAME_CHECKS`* control this behavior. If the coordinate variables are one-dimensional, indicating a possibly regular grid, GDAL will then check whether the coordinate variables values are equally-spaced and, if applicable, calculate a geotransform. Starting with GDAL 3.11, if the `GeoTransform` metadata is also present, the values of the calculated and stored geotransforms will be compared and an warning raised if they do not correspond. In such a case, preference will be given to the calculated values in GDAL 3.12.3 and above, or the stored values in earlier versions. (A significant mismatch may arise if a file is created by GDAL, modified using another software package that ignores the `GeoTransform` attribute, and then read in again by GDAL.)

An alternative method of storing the geotransform relies on storing the spatial extent of the dataset using four attributes on the grid mapping variable: `Northermost_Northing`, `Southernmost_Northing`, `Eastermost_Easting`, and `Westernmost_Eastering`. If these attributes are present (and `GeoTransform` is not), they will be used instead of the computed values.

6.102.5 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are available:

- **LIST_ALL_ARRAYS=[YES/NO]:** (GDAL >= 3.11) Defaults to `NO`. In classic 2D mode, whereas the subdataset list should include all arrays, including those with 1 dimension.
- **HONOUR_VALID_RANGE=[YES/NO]:** Defaults to `YES`. Whether to set to nodata pixel values outside of the validity range indicated by valid_min, valid_max or valid_range attributes.
- **IGNORE_XY_AXIS_NAME_CHECKS=[YES/NO]:** (GDAL >= 3.4.2) Defaults to `NO`. Whether X/Y dimensions should be always considered as geospatial axis, even if the lack conventional attributes confirming it.
- **VARIABLES_AS_BANDS=[YES/NO]:** (GDAL >= 3.5) Defaults to `NO`. If set to YES, and if the netCDF file only contains 2D variables of the same type and indexed by the same dimensions, then they should be reported as multiple bands of a same dataset. Default is NO (that is each variable will be reported as a separate subdataset)
- **ASSUME_LONGLAT=[YES/NO]:** (GDAL >= 3.7) Defaults to `NO`. Whether a Geographic CRS (OGC:CRS84) should be assumed and applied when, none has otherwise been found, a meaningful geotransform has been found, and that geotransform is within the bounds -180,360 -90,90.
- **PRESERVE_AXIS_UNIT_IN_CRS=[YES/NO]:** (GDAL >= 3.8) Defaults to `NO`. Whether unusual linear axis unit (km) should be kept as such, instead of being normalized to metre. The default is NO, ie that are being normalized to metre (previous GDAL versions kept the original unit)

6.102.6 Creation Issues

This driver supports creation of NetCDF file following the CF-1 convention. You may create set of 2D datasets. Each variable array is named Band1, Band2, ... BandN.

Each band will have metadata tied to it giving a short description of the data it contains.

6.102.7 GDAL NetCDF Metadata

All NetCDF attributes are transparently translated as GDAL metadata.

The translation follow these directives:

- Global NetCDF metadata have a **NC_GLOBAL** tag prefixed.
- Dataset metadata have their **variable name** prefixed.
- Each prefix is followed by a **#** sign.
- The NetCDF attribute follows the form: **name=value**.

Example:

```
$ gdalinfo NETCDF:"sst.nc":tos
Driver: netCDF/Network Common Data Format
Size is 180, 170
Coordinate System is `'
Origin = (1.000000,-79.500000)
Pixel Size = (1.98888889,0.99411765)
Metadata:
```

NetCDF global attributes

```
NC_GLOBAL#title=IPSL  model output prepared for IPCC Fourth Assessment SRES A2 experiment
```

Variables attributes for: tos, lon, lat and time

```
tos#standard_name=sea_surface_temperature
tos#long_name=Sea Surface Temperature
tos#units=K
tos#cell_methods=time: mean (interval: 30 minutes)
tos#_FillValue=1.000000e+20
tos#missing_value=1.000000e+20
tos#original_name=sosstsst
tos#original_units=degC
tos#history= At   16:37:23 on 01/11/2005: CMOR altered the data in the following ways:␣
↪added 2.73150E+02 to yield output units;  Cyclical dimension was output starting at a␣
↪different lon;
lon#standard_name=longitude
lon#long_name=longitude
lon#units=degrees_east
lon#axis=X
lon#bounds=lon_bnds
lon#original_units=degrees_east
lat#standard_name=latitude
lat#long_name=latitude
lat#units=degrees_north
lat#axis=Y
lat#bounds=lat_bnds
lat#original_units=degrees_north
time#standard_name=time
time#long_name=time
time#units=days since 2001-1-1
time#axis=T
time#calendar=360_day
time#bounds=time_bnds
time#original_units=seconds since 2001-1-1
```

On writing, when using the CreateCopy() interface or gdal_translate, dataset level metadata that follows the naming convention NC_GLOBAL#key=value will be used to write the netCDF attributes. Metadata set at the band level using key=value will also be used to write variable attributes.

6.102.8 Product specific behavior

6.102.8.1 Sentinel 5

Added in version 3.4.

The most verbose metadata is reported in the `json:ISO_METADATA`, `json:ESA_METADATA`, `json:EOP_METADATA`, `json:QA_STATISTICS`, `json:GRANULE_DESCRIPTION`, `json:ALGORITHM_SETTINGS` and `json:SUPPORT_DATA` metadata domains.

Can be discovered for example with:

```
gdalinfo -mdd all -json S5P.nc
```

6.102.9 Creation Options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following creation options are available:

- **FORMAT=[NC/NC2/NC4/NC4C]:** Defaults to `NC`. Set the NetCDF file format to use. NC2 is normally supported by recent NetCDF installations, but NC4 and NC4C are available if NetCDF was compiled with NetCDF-4 (and HDF5) support.
- **BAND_NAMES=value:** (GDAL >= 3.9.0) Defaults to `Band1,Band2,...`. A comma-separated list of band names.
- **COMPRESS=[NONE/DEFLATE]:** Set the compression to use. DEFLATE is only available if NetCDF has been compiled with NetCDF-4 support. NC4C format is the default if DEFLATE compression is used.
- **ZLEVEL=1-9:** Defaults to `1`. Set the level of compression when using DEFLATE compression. A value of 9 is best, and 1 is least compression. The default is 1, which offers the best time/compression ratio.
- **WRITE_BOTTOMUP=[YES/NO]:** Defaults to `YES`. Set the y-axis order for export, overriding the order detected by the driver. NetCDF files are usually assumed "bottom-up", contrary to GDAL's model which is "north up". This normally does not create a problem in the y-axis order, unless there is no y axis geo-referencing. The default for this setting is YES, so files will be exported in the NetCDF default "bottom-up" order. For import see *`GDAL_NETCDF_BOTTOMUP`* below.
- **WRITE_GDAL_TAGS=[YES/NO]:** Define if GDAL tags used for georeferencing (spatial_ref and GeoTransform) should be exported, in addition to CF tags. Not all information is stored in the CF tags (such as named datums and EPSG codes), therefore the driver exports these variables by default. In import the CF "grid_mapping" variable takes precedence and the GDAL tags are used if they do not conflict with CF metadata. In GDAL 4, spatial_ref will not be exported. The crs_wkt CF metadata attribute will be used instead.
- **WRITE_LONLAT=[YES/NO/IF_NEEDED]:** Define if CF lon/lat variables are written to file. Default is YES for geographic SRS and NO for projected SRS. This is normally not necessary for projected SRS as GDAL and many applications use the X/Y dimension variables and CF projection information. Use of IF_NEEDED option creates lon/lat variables if the projection is not part of the CF-1.5 standard.
- **TYPE_LONLAT=[float/double]:** Set the variable type to use for lon/lat variables. Default is double for geographic SRS and float for projected SRS. If lon/lat variables are written for a projected SRS, the file is considerably large (each variable uses X*Y space), therefore *`TYPE_LONLAT=float`* and *`COMPRESS=DEFLATE`* are advisable in order to save space.
- **PIXELTYPE=[DEFAULT/SIGNEDBYTE]:** By setting this to SIGNEDBYTE, a new Byte file can be forced to be written as signed byte. Starting with GDAL 3.7, this option is deprecated and Int8 should rather be used.
- **WRITE_GDAL_VERSION=[YES/NO]:** (GDAL >= 3.5.0) Defaults to `YES`. Define if a "GDAL" text global attribute should be added on file creation with the GDAL version
- **WRITE_GDAL_HISTORY=[YES/NO]:** (GDAL >= 3.5.0) Defaults to `YES`. Define if the "history" global attribute should be prepended with a date/time and GDAL information.

6.102.10 Creation of multidimensional files with CreateCopy() 2D raster API

Starting with GDAL 3.1, the preferred way of creating > 2D files is to use the the *Multidimensional Raster Data Model* API. However it is possible to create such files with the 2D raster API using the CreateCopy() method (note that at time of writing, this is not supported using the Create() method).

The `NETCDF_DIM_EXTRA={dim1_name,...dimN_name}` metadata item must be set on the source dataset, where dim1_name is the name of the slowest varying dimension and dimN_name the name of the fastest varying one.

For each extra dimension, the `NETCDF_DIM_{dim_name}_DEF={dimension_size,netcdf_data_type}` metadata item must be set where dimension_size is the size of the dimension (number of samples along that dimension) and netcdf_data_type is the integer value for the netCDF data type of the corresponding indexing variable. Among the most useful data types:

- 4 for Int
- 5 for Float

- 6 for Double
- 10 for Int64

The `NETCDF_DIM_{dim_name}_VALUES={value1,...valueN}` is set to define the values of the indexing variable corresponding to dimension.

`dim_name#attribute` metadata items can also be set to define the attributes of the indexing variable of the dimension.

Example of creation of a Time,Z,Y,X 4D file in Python:

```
# Create in-memory file with required metadata to define the extra >2D
# dimensions
size_z = 2
size_time = 3
src_ds = gdal.GetDriverByName('MEM').Create('', 4, 3, size_z * size_time)
src_ds.SetMetadataItem('NETCDF_DIM_EXTRA', '{time,Z}')
# 6 is NC_DOUBLE
src_ds.SetMetadataItem('NETCDF_DIM_Z_DEF', f"{{{size_z},6}}")
src_ds.SetMetadataItem('NETCDF_DIM_Z_VALUES', '{1.25,2.50}')
src_ds.SetMetadataItem('Z#axis', 'Z')
src_ds.SetMetadataItem('NETCDF_DIM_time_DEF', f"{{{size_time},6}}")
src_ds.SetMetadataItem('NETCDF_DIM_time_VALUES', '{1,2,3}')
src_ds.SetMetadataItem('time#axis', 'T')
src_ds.SetGeoTransform([2,1,0,49,0,-1])

# Create netCDF file
gdal.GetDriverByName('netCDF').CreateCopy('out.nc', src_ds)
```

6.102.11 Configuration Options

- **GDAL_NETCDF_BOTTOMUP=[YES/NO]:** Set the y-axis order for import, overriding the order detected by the driver. This option is usually not needed unless a specific dataset is causing problems (which should be reported on GitHub).
- **GDAL_NETCDF_VERIFY_DIMS=[YES/STRICT]:** Defaults to `YES`. Try to guess which dimensions represent the latitude and longitude only by their attributes (STRICT) or also by guessing the name (YES).
- **GDAL_NETCDF_IGNORE_XY_AXIS_NAME_CHECKS=[YES/NO]:** Defaults to `NO`. Whether X/Y dimensions should be always considered as geospatial axis, even if the lack conventional attributes confirming it.
- **GDAL_NETCDF_ASSUME_LONGLAT=[YES/NO]:** (GDAL >= 3.7) Defaults to `NO`. Whether a Geographic CRS (OGC:CRS84) should be assumed and applied when none has otherwise been found, a meaningful geotransform has been found, and that geotransform is within the bounds -180,360 -90,90. Equivalent to open option *ASSUME_LONGLAT*.
- **GDAL_NETCDF_REPORT_EXTRA_DIM_VALUES=[YES/NO]:** (GDAL >= 3.10.1) Defaults to `NO`. For a netCDF dataset stored on a remote file system (`/vsicurl/`, `/vsis3/`), getting the content of the `NETCDF_DIM_{dim_name}_VALUES` metadata item can be a slow operation when the dimension is unlimited. It is thus disabled by default for such remote files. By setting this configuration option to YES, you force GDAL to get the content of such metadata items.

6.102.12 VSI Virtual File System API support

With Linux kernel >=4.3 and libnetcdf >=4.5, read operations on /vsi file systems are supported using the userfaultfd Linux system call. If running from a container, that system call may be unavailable by default. For example with Docker, `--security-opt seccomp=unconfined` might be needed.

Corollary: operations on /vsi file systems are *not* supported on Windows or MacOSX. If the netCDF file is a NetCDF 4 / HDF5 file, and the HDF5 driver is available, you may set the *GDAL_SKIP* configuration option to `netCDF` to force the use of the HDF5 driver. Note that specificities of the netCDF driver, such as support georeferencing with the netCDF CF conventions, will not be available.

6.102.13 NetCDF-4 groups support on reading (GDAL >= 3.0)

The driver has undergone significant changes in GDAL 3.0 to support NetCDF-4 groups on reading:

- Explore recursively all nested groups to create the subdatasets list
- Subdatasets in nested groups use the /group1/group2/.../groupn/var standard NetCDF-4 convention, except for variables in the root group which do not have a leading slash for backward compatibility with the NetCDF-3 driver
- Global attributes of each nested group are also collected in the GDAL dataset metadata, using the same convention /group1/group2/.../groupn/NC_GLOBAL#attr_name, except for the root group which do not have a leading slash for backward compatibility
- When searching for a variable containing auxiliary information on the selected subdataset, like coordinate variables or grid_mapping, we now also search in parent groups and their children as specified in Support of groups in CF

6.102.14 Multidimensional API support

Added in version 3.1.

The netCDF driver supports the *Multidimensional Raster Data Model* for reading and creation operations.

The *GDALGroup::GetGroupNames()* method supports the following options:

- GROUP_BY=SAME_DIMENSION. If set, single-dimensional variables will be exposed as a "virtual" subgroup. This enables the user to get a clearer organization of variables, for example in datasets where variables belonging to different trajectories are indexed by different dimensions but mixed in the same netCDF group.

The *GDALGroup::OpenGroup()* method supports the following options:

- GROUP_BY=SAME_DIMENSION. See above description

The *GDALGroup::GetMDArrayNames()* method supports the following options:

- SHOW_ALL=YES/NO. Defaults to NO. If set to YES, all variables will be listed.
- SHOW_ZERO_DIM=YES/NO. Defaults to NO. If set to NO, variables with 0-dimension will not be listed.
- SHOW_COORDINATES=YES/NO. Defaults to YES. If set to NO, variables referenced in the `coordinates` attribute of another variable will not be listed.
- SHOW_BOUNDS=YES/NO. Defaults to YES. If set to NO, variables referenced in the `bounds` attribute of another variable will not be listed.
- SHOW_INDEXING=YES/NO. Defaults to YES. If set to NO, single-dimensional variables whose name is equal to the name of their indexing variable will not be listed.
- SHOW_TIME=YES/NO. Defaults to YES. If set to NO, single-dimensional variables whose `standard_name` attribute is "time" will not be listed.
- GROUP_BY=SAME_DIMENSION. If set, single-dimensional variables will not be listed

The *GDALGroup::OpenMDArray()* method supports the following options:

- USE_DEFAULT_FILL_AS_NODATA=YES/NO. (GDAL >= 3.6) Defaults to NO. If set to YES, the default fill value will be used as nodata when there is no _FillValue or missing_value attribute (except on variables of type Byte, UByte, Char)
- RAW_DATA_CHUNK_CACHE_SIZE=<integer>. (GDAL >= 3.10, advanced libnetcdf parameter) The total size of the libnetcdf raw data chunk cache, in bytes. Default value (at least for some versions of libnetcdf) is 1 MB. Only for netCDF4/HDF5 files.
- CHUNK_SLOTS=<integer>. (GDAL >= 3.10, advanced libnetcdf parameter) The total size of the libnetcdf raw data chunk cache, in bytes. Default value (at least for some versions of libnetcdf) is 521. Only for netCDF4/HDF5 files.
- PREEMPTION=<float> between 0 and 1. (GDAL >= 3.10, advanced libnetcdf parameter) Indicates how much chunks from libnetcdf chunk cache that have been fully read are favored for preemption. A value of zero means fully read chunks are treated no differently than other chunks (the preemption is strictly least-recently used) while a value of one means fully read chunks are always preempted before other chunks. Default value (at least for some versions of libnetcdf) is 0.75. Only for netCDF4/HDF5 files.

For RAW_DATA_CHUNK_CACHE_SIZE, CHUNK_SLOTS and PREEMPTION, consult nc_set_var_chunk_cache and documentation about netCDF chunk cacke

The *GDALGroup::CreateMDArray()* method supports the following options:

- NC_TYPE=NC_CHAR/NC_BYTE/NC_INT64/NC_UINT64: to overload the netCDF data type normally deduced from the GDAL data type passed to CreateMDArray(). NC_CHAR can only be used for strings of a fixed size.
- BLOCKSIZE=size_dim0,size_dim1,...,size_dimN: to set the netCDF chunk size, as set by nc_def_var_chunking(). There must be exactly as many values as the number of dimensions passed to CreateMDArray()
- COMPRESS=DEFLATE: to ask for deflate compression
- ZLEVEL=number: DEFLATE compression level (1-9)
- CHECKSUM=YES/NO: Whether to turn on Fletcher32 checksums. Checksum generation requires chunking, and if no explicit chunking has been asked with the BLOCKSIZE option, a default one will be used. Defaults to NO.
- FILTER=filterid,param1,...,paramN: Define a filter (typically a compression method) used for writing. This should be a list of numeric values, separated by commas. The first value is the filter id (list of potential values at https://support.hdfgroup.org/services/contributions.html#filters) and following values are per-filter parameters. More details about netCDF-4 filter support at https://www.unidata.ucar.edu/software/netcdf/docs/md__Users_wfisher_Desktop_docs_netcdf-c_docs_filters.html

6.102.15 Driver building

This driver is compiled with the UNIDATA NetCDF library.

You need to download or compile the NetCDF library before configuring GDAL with NetCDF support.

6.102.16 See Also:

- *Vector side of the netCDF driver.*
- NetCDF CF-1.5 convention
- NetCDF compiled libraries
- NetCDF Documentation

6.103 NGSGEOID - NOAA NGS Geoid Height Grids

Driver short name

NGSGEOID

Driver built-in by default

This driver is built-in by default

GDAL supports reading NOAA NGS geoid height grids in binary format (.bin files). Those files can be used for vertical datum transformations.

6.103.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.103.2 See also

- Description of NGS Geoid Height Grids format
- GEOID09 main page
- USGG2009 main page

6.104 NGW -- NextGIS Web

Driver short name

NGW

Build dependencies

libcurl

NextGIS Web - is a server GIS, which allows storing and editing geodata and to display maps in web browser. Also NextGIS Web can share geodata with other NextGIS software.

NextGIS Web has the following features:

- Display maps in a web browser (different maps with different layers and styles)

- Flexible permissions management
- Load geodata from PostGIS or import from GIS formats (ESRI Shape, GeoJSON or GeoTIFF)
- Load vector geodata in the following formats: GeoJSON, CSV, ESRI Shape, Mapinfo tab
- Import map styles from QGIS project or set them manually
- Act as a server for TMS, WMS, MVT, WFS
- Act as a client for WMS
- User can add photos to records, change record attributes via web interface or WFS-T protocol

NextGIS Web - is an open source software (license GPL v2+, see GNU General Public License, version 2).

6.104.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

6.104.2 Driver

NextGIS Web supports several raster types:

- Raster style
- Vector style
- WMS layer
- WMS Service
- Web map as combination of raster and vector styles
- Base map resource
- QML Raster style
- QML Vector style
- Raster layer
- Raster mosaic
- Tileset

Each NextGIS Web raster layer can have one or more raster styles. Each NextGIS Web vector or PostGIS layer can have one or more vector styles (QGIS qml or MapServer xml). WMS layers from external WMS service have no styles. WMS Service is usual WMS protocol implementation.

NGW driver supports raster, vector, QML raster, QML vector styles, web map, base map layer, WMS layers and raster layer as COG. You can get raster data as tiles or image or COG (only tiles and COG are supported now).

The driver supports read and copy from existing source dataset operations on rasters.

6.104.3 Dataset name syntax

The minimal syntax to open a NGW datasource is: NGW:[NextGIS Web URL][/resource/][resource identifier]

- **NextGIS Web URL** may be an url to nextgis.com cloud service (for example, https://demo.nextgis.com), or some other url including port and additional path (for example, http://192.168.1.1:8000/test).

- **resource** is mandatory keyword dividing resource identifier from the rest of URL.
- **resource identifier** this is positive number from 0 and above. This may be a resource group, vector, PostGIS or raster layer, style.

If identifier is resource group, all vector layers, PostGIS, raster layers, styles will be listed as child resources. In other case this will be a separate raster.

6.104.4 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The following configuration options are available:

- **NGW_USERPWD=value:** User name and password separated with colon. Optional and can be set using open options.
- **NGW_CACHE_EXPIRES=<seconds>:** Defaults to `604800` (7 days). Time in seconds cached files will stay valid. If cached file expires it is deleted when maximum size of cache is reached. Also expired file can be overwritten by the new one from web.
- **NGW_CACHE_MAX_SIZE=<bytes>:** Defaults to `67108864` (64 Mb). The cache maximum size in bytes. If cache reached maximum size, expired cached files will be deleted.
- **NGW_JSON_DEPTH=value:** The depth of json response that can be parsed. If depth is greater than this value, parse error occurs.
- **NGW_CONNECTTIMEOUT=value:** Maximum delay for the connection to be established before being aborted in seconds.
- **NGW_TIMEOUT=value:** Maximum delay for the whole request to complete before being aborted in seconds.
- **NGW_MAX_RETRY=value:** Maximum number of retry attempts if a 429, 502, 503 or 504 HTTP error occurs.
- **NGW_RETRY_DELAY=value:** Number of seconds between retry attempts.

6.104.5 Authentication

Any operations (read, write, get metadata, change properties, etc.) may require an authenticated access. Authenticated access is obtained by specifying user name and password in open, create or configuration options.

6.104.6 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are available:

- **USERPWD=value:** Username and password, separated by colon.
- **CACHE_EXPIRES=<seconds>:** Defaults to `604800` (7 days). Time in seconds cached files will stay valid. If cached file expires it is deleted when maximum size of cache is reached. Also expired file can be overwritten by the new one from web.
- **CACHE_MAX_SIZE=<bytes>:** Defaults to `67108864` (64 Mb). The cache maximum size in bytes. If cache reached maximum size, expired cached files will be deleted.
- **JSON_DEPTH=value:** Defaults to `32`. The depth of json response that can be parsed. If depth is greater than this value, parse error occurs.
- **CONNECTTIMEOUT=value:** Maximum delay for the connection to be established before being aborted in seconds.
- **TIMEOUT=value:** Maximum delay for the whole request to complete before being aborted in seconds.

- **MAX_RETRY=value:** Maximum number of retry attempts if a 429, 502, 503 or 504 HTTP error occurs.
- **RETRY_DELAY=value:** Number of seconds between retry attempts.

6.104.7 Create copy options

NextGIS Web supports only GeoTIFF file format. Prior version 3.1 supported only 3 (RGB) or 4 (RGBA) bands rasters with datatype Byte. In CreateCopy function if source dataset has GeoTIFF file format it will copy as is. For other formats the additional transformation to temporary GeoTIFF file will execute.

The following copy options are available:

- KEY - Key value. Must be unique in whole NextGIS Web instance. Optional.
- DESCRIPTION - Resource description. Optional.
- RASTER_STYLE_NAME - Raster style name. Optional. Default is same as raster layer name.
- RASTER_QML_PATH - Path to QGIS QML raster style file. Optional for RGB/RGBA, for other bands count/pixel types is mandatory.
- USERPWD - Username and password, separated by colon.
- CACHE_EXPIRES=604800 - Time in seconds cached files will stay valid. If cached file expires it is deleted when maximum size of cache is reached. Also expired file can be overwritten by the new one from web. Defaults to 604800 (7 days).
- CACHE_MAX_SIZE=67108864 - The cache maximum size in bytes. If cache reached maximum size, expired cached files will be deleted. Defaults to 67108864 (64Mb).
- JSON_DEPTH=32 - The depth of json response that can be parsed. If depth is greater than this value, parse error occurs.

6.104.8 Metadata

NextGIS Web metadata are supported in datasource, vector, PostGIS, raster layers and styles. Metadata are stored at specific domain "NGW". NextGIS Web supported metadata are strings and numbers. Metadata keys with decimal numbers will have suffix **.d** and for real numbers - **.f**. To create new metadata item, add new key=value pair in NGW domain use the *SetMetadataItem* function and appropriate suffix. During transferring to NextGIS Web, suffix will be omitted. You must ensure that numbers correctly transform from string to number.

Resource description and key map to appropriate *description* and *keyname* metadata items in default domain. Changing those metadata items will cause an update of resource properties.

Resource creation date, type and parent identifier map to appropriate read-only metadata items *creation_date*, *resource_type* and *parent_id* in default domain.

6.104.9 Examples

Read datasource contents (1730 is resource group identifier):

```
gdalinfo NGW:https://demo.nextgis.com/resource/1730
```

Read raster details (1734 is raster layer identifier):

```
gdalinfo NGW:https://demo.nextgis.com/resource/1734
```

6.104.10 See also

- *Vector side of the driver*
- NextGIS Web documentation
- NextGIS Web for developers

6.105 NITF -- National Imagery Transmission Format (also CIB, CADRG, ECRG, HRE)

Driver short name

NITF

Driver built-in by default

This driver is built-in by default

6.105.1 NITF -- Advanced Driver Information

The NITF (National Imagery Transmission Format) driver in GDAL includes a number of advanced, and somewhat esoteric options not suitable for the *general end user documentation* for the driver. This information is collected here, and is primarily aimed at developers and advanced users.

6.105.1.1 CGM Segments

NITF files that have CGM data (that is segment type GR - graphics, or SY with an STYPE value of 'C') will make that information available as metadata in the CGM domain. The returned metadata will look something like:

```
SEGMENT_COUNT=1
SEGMENT_0_SLOC_ROW=25
SEGMENT_0_SLOC_COL=25
SEGMENT_0_SDLVL=2
SEGMENT_0_SALVL=1
SEGMENT_0_CCS_ROW=00025
SEGMENT_0_CCS_COL=00025
SEGMENT_0_DATA=\0!\0...
```

The SLOC_ROW and SLOC_COL values are the placement of the CGM object relative to the base (SALVL) image. The CCS_ROW/COL values are relative to the common coordinate system. The _SDLVL is the display level. The DATA is the raw CGM data with "backslash quotable" escaping applied. All occurrences of ASCII zero will be translated to '0', and all backslashes and double quotes will be backslashed escaped. The CPLUnescapeString() function can be used to unescape the data into binary format using scheme CPLES_BackslashQuotable.

To add CGM data to a NITF image, you can pass creation options in the following format:

```
CGM=SEGMENT_COUNT=1
CGM=SEGMENT_0_SLOC_ROW=25
CGM=SEGMENT_0_SLOC_COL=25
CGM=SEGMENT_0_SDLVL=2
CGM=SEGMENT_0_SALVL=1
CGM=SEGMENT_0_DATA=\0!\0...
```

Notice that passing CGM as creation options will overwrite existing CGM segment read in the CGM metadata domain.

While GDAL does not support parsing or rendering CGM data, at least one user has found the UniConverter library useful for this purpose.

6.105.1.2 Multi-Image NITF Files

NITF files with more than one image segment (IM) will present the image segments as subdatasets. Opening a multiple NITF file by filename will provide access to the first image segment. The subdataset metadata for a 3 image NITF file might look like:

```
Subdatasets:
  SUBDATASET_1_NAME=NITF_IM:0:multi_image_jpeg_2.0.ntf
  SUBDATASET_1_DESC=Image 1 of multi_image_jpeg_2.0.ntf
  SUBDATASET_2_NAME=NITF_IM:1:multi_image_jpeg_2.0.ntf
  SUBDATASET_2_DESC=Image 2 of multi_image_jpeg_2.0.ntf
  SUBDATASET_3_NAME=NITF_IM:2:multi_image_jpeg_2.0.ntf
  SUBDATASET_3_DESC=Image 3 of multi_image_jpeg_2.0.ntf
```

In this case opening "multi_image_jpeg_2.0.ntf" directly will give access to "NITF_IM:0:multi_image_jpeg_2.0.ntf". To open the others use the corresponding subdataset names. The Subdataset mechanism is generic GDAL concept discussed in the *Raster Data Model* document.

6.105.1.3 Text Segments

NITF files that have text segments (that is segment type TX) will make that information available as metadata in the TEXT domain. The returned metadata will look something like:

```
HEADER_0=TE       00020021216151629xxxxxxxxxxxxxxxxxxxxxxxxxxx
DATA_0=This is test text file 01.

HEADER_1=TE       00020021216151629xxxxxxxxxxxxxxxxxxxxxxxxxxx
DATA_1=This is test text file 02.

HEADER_2=TE       00020021216151629xxxxxxxxxxxxxxxxxxxxxxxxxxx
DATA_2=This is test text file 03.

HEADER_3=TE       00020021216151629xxxxxxxxxxxxxxxxxxxxxxxxxxx
DATA_3=This is test text file 04.

HEADER_4=TE       00020021216151629xxxxxxxxxxxxxxxxxxxxxxxxxxx
DATA_4=This is test text file 05.
```

The argument to DATA_n is the raw text of the n'th (zero based) text segment with no escaping of any kind applied.

The TEXT segment header data is preserved in HEADER_n metadata item. The CreateCopy() method on the NITF driver also supports creating text segments on the output file as long as the input file has metadata in the TEXT domain as defined above.

To add TEXT data to a NITF image, you can also pass creation options in the following format:

```
TEXT=HEADER_0=TE       00020021216151629xxxxxxxxxxxxxxxxxxxxxxxxxxx
TEXT=DATA_0=This is test text file 01.
TEXT=HEADER_1=TE       00020021216151629xxxxxxxxxxxxxxxxxxxxxxxxxxx
TEXT=DATA_1=This is test text file 02.
```

Notice that passing TEXT as creation options will overwrite existing text segment read in the TEXT metadata domain.

6.105.1.4 TREs

NITF files with registered (or unregistered?) extensions on the file header, or the referenced image header will make them available in a raw form in metadata via the TRE domain. The TRE domain will hold one metadata item per TRE which will have the name of the TRE as the name, and the data of the TRE as the contents. The data contents will be "backslash escaped" like CGM data above.

In case of multiple occurrences of the same TRE, the second occurrence will be named "TRENAME_2", the third "TRENAME_3" where TRENAME is the TRE name.

```
Metadata (TRE):
  GEOPSB=MAPM  World Geodetic System 1984
               WGE World Geodetic System 1984
                   WE Geodetic
                      GEODMean Sea
                          MSL 000000000000000
                                                  0000
  PRJPSB=Polar Stereographic
         PG2-00090.00000250000039.99999884000000000000000000000000000000
  MAPLOB=M  0598005958-000003067885.8-000002163353.8
```

6.105.1.5 TREs as xml:TRE

All TREs found in file and matching one of the TRE description of the nitf_spec.xml in GDAL data directory will be reported as XML content in the xml:TRE metadata domain.

```
Metadata (xml:TRE):
<tres>
  <tre name="RSMDCA" location="des TRE_OVERFLOW">
    <field name="IID" value="2_8" />
    <field name="EDITION" value="1101222272-2" />
    <field name="TID" value="1101222272-1" />
    <field name="NPAR" value="06" />
    <field name="NIMGE" value="001" />
    <field name="NPART" value="00006" />
    <repeated name="IMAGE" number="1">
      <group index="0">
        <field name="IID" value="2_8" />
        <field name="NPARI" value="06" />
      </group>
    </repeated>
    <field name="XUOL" value="-2.42965895449297E+06" />
    <field name="YUOL" value="-4.76049894293300E+06" />
    <field name="ZUOL" value="+3.46898407315533E+06" />
    <field name="XUXL" value="+8.90698769551156E-01" />
    <field name="XUYL" value="+2.48664813021570E-01" />
    <field name="XUZL" value="-3.80554217799520E-01" />
    <field name="YUXL" value="-4.54593996792805E-01" />
    <field name="YUYL" value="+4.87215943350720E-01" />
    <field name="YUZL" value="-7.45630553709282E-01" />
    <field name="ZUXL" value="+0.00000000000000E+00" />
    <field name="ZUYL" value="+8.37129879594448E-01" />
    <field name="ZUZL" value="+5.47004172461403E-01" />
[...]
```

(continues on next page)

(continued from previous page)

```
    <repeated name="DERCOV" number="21">
      <group index="0">
        <field name="DERCOV" value="+5.77388827727787E+04" />
      </group>
[...]
      <group index="20">
        <field name="DERCOV" value="+1.14369570920252E-02" />
      </group>
    </repeated>
  </tre>
  <tre name="RSMECA" location="des TRE_OVERFLOW">
[...]
  </tre>
  <tre name="RSMIDA" location="des TRE_OVERFLOW">
[...]
  </tre>
  <tre name="RSMPCA" location="des TRE_OVERFLOW">
[...]
  </tre>
</tres>
```

6.105.1.6 TRE creation from hexadecimal data

TRE data can be added to a newly created NITF file in hexadecimal format to encode binary data such as unsigned int or floating point types. The hexadecimal TRE creation option is supplied as "TRE=HEX/<tre_name>=<hex_tre_data>" or "FILE_TRE=HEX/<tre_name>=<hex_tre_data>

```
# Encode "ABC" as 3 bytes of hex data, "414243"
ds = gdal.GetDriverByName('NITF').Create('/vsimem/file.ntf', 1, 1, options=["TRE=HEX/
↪TSTTRE=414243"])
```

6.105.1.7 TRE creation as TRE_OVERFLOW DES

The following example shows how to create a CSEPHA TRE as a TRE_OVERFLOW DES:

```
DESOFLW = "IXSHD "
DESITEM = "001" # index (starting at 1) of the image to which this TRE_OVERFLOW applies␣
↪too
DESSHL = "0000"
des_header = "02U" + " " * 166 + DESOFLW + DESITEM + DESSHL

# Totally dummy content for CSEPHA data
EPHEM_FLAG = " " * 12
DT_EPHEM = " " * 5
DATE_EPHEM = " " * 8
T0_EPHEM = " " * 13
NUM_EPHEM = "001"
EPHEM_DATA = " " * (1 * (3 * 12))
CSEPHA_DATA = EPHEM_FLAG + DT_EPHEM + DATE_EPHEM + T0_EPHEM + NUM_EPHEM + EPHEM_DATA

des_data = "CSEPHA" + ("%05d" % len(CSEPHA_DATA)) + CSEPHA_DATA
des = des_header + des_data
```

(continues on next page)

(continued from previous page)

```
ds = gdal.GetDriverByName("NITF").Create(
    "/vsimem/nitf_DES.ntf",
    1,
    1,
    options=["RESERVE_SPACE_FOR_TRE_OVERFLOW=YES",
             "DES=TRE_OVERFLOW=" + des],
)
ds = None
```

Note the use of the RESERVE_SPACE_FOR_TRE_OVERFLOW=YES creation option to reserve space for the IXSOFL field in the image subheader pointed by DESITEM.

6.105.1.8 Data Extension Segments (xml:DES)

Data Extension Segments (DESs) are user-defined metadata extensions to the NITF format. The DES metadata is available through the xml:DES metadata domain. The xml:DES domain returns an XML string with standard and user-defined DES fields in plain text, and the user-defined DES data as Base64 encoded text. The following is an example XML structure:

```
<des_list>
  <des name="TEST">
    <field name="NITF_DESVER" value="02" />
    <field name="NITF_DECLAS" value="U" />
    <field name="NITF_DESCLSY" value="" />
    [...]
    <field name="NITF_DESCTLN" value="" />
    <field name="NITF_DESSHL" value="0004" />
    <field name="NITF_DESSHF" value="ABCD" />
    <field name="NITF_DESDATA" value="MTIzNDU2Nzg5MA==" />
  </des>
</des_list>
```

6.105.1.9 Raw File / Image Headers

In some cases application may need to recover very specific information from the image or file headers that isn't normally available as metadata. In this case it is possible to query the "NITF_METADATA" metadata domain. The complete file and image headers will be returned as metadata in base64 encoded format. Something like:

```
Metadata (NITF_METADATA):
  NITFFileHeader=002213 TklURjAyLjAwMDEgICAgVTIxN0cwSjA...
  NITFImageSubheader=439 SU1NaXNzaW5nIElEMjUxNTI1NTlaTU...
```

Note that the ascii encoded numeric values prefixing the base64 encoded header is the length (decoded) in bytes, followed by one space.

6.105.1.10 Writing multiple Image Segments

(GDAL >= 3.4)

Writing a NITF file with several image segments is done with the `APPEND_SUBDATASET=YES` creation option but requires some care.

The file must be created with the content of the first image segment with the `NUMI` creation option to specify the total number of image segments and optionally with the `NUMDES` creation option with the number of DE segments. TREs

that apply to the whole file should also be specified at that stage. The `APPEND_SUBDATASET` creation option must NOT be specified.

Subsequent images are appended with the `APPEND_SUBDATASET=YES` creation option. The `IDLVL`, `IALVL`, `ILOCROW` and `ILOCCOL` creation option may be specified.

On the last image segment, the DES content, if any must be specified with the `DES` creation option.

> **Note**
>
> The file will only be valid if all NUMI images have been actually written.

Example:

```
gdal_translate first_image.tif  dest.tif -co NUMI=3 -co NUMDES=1
gdal_translate second_image.tif dest.tif -co APPEND_SUBDATASET=YES -co IC=C3 -co IDLVL=2
gdal_translate third_image.tif  dest.tif -co APPEND_SUBDATASET=YES -co IC=C8 -co IDLVL=3␣
↪-co "DES=DES1={des_content}"
```

6.105.1.11 Reading SAR products

Starting with GDAL 3.11, NITF products whose `ICAT` field is set to `SAR` and which have bands with `I` (in-phase) and `Q` (quadrature) `ISUBCAT` values will be represented as a single-band of complex type where in-phase is the real part and quadrature the imaginary part.

GDAL supports reading of several subtypes of NITF (National Imagery Transmission Format) image files, and writing NITF 2.1 files, and limited writing support for NITF 2.0. NITF 1.1, NITF 2.0, NITF 2.1 and NSIF 1.0 files with uncompressed, ARIDPCM (Adaptive Recursive Interpolated Differential Pulse Code Modulation), JPEG compressed, JPEG2000 (with Kakadu, ECW SDKs or other JPEG2000 capable driver) or VQ (Vector Quantized) compressed images should be readable.

The read support test has been tested on various products, including CIB (Controlled Image Base) and CADRG (Compressed ARC Digitized Raster Graphics) frames from RPF (Raster Product Format) products, ECRG (Enhanced Compressed Raster Graphics) frames, HRE (High Resolution Elevation) products. Write support for CADRG is available since GDAL 3.13.

Color tables for pseudocolored images are read. In some cases nodata values may be identified.

Lat/Long extents are read from the IGEOLO (Image GeoLocation) information in the image header if available. If high precision lat/long georeferencing information is available in RPF auxiliary data it will be used in preference to the low precision IGEOLO information. In case a BLOCKA (Image Block Information) instance is found, the higher precision coordinates of BLOCKA are used if the block data covers the complete image - that is the L_LINES field with the row count for that block is equal to the row count of the image. Additionally, all BLOCKA instances are returned as metadata. If GeoSDE TRE (Tagged Record Extension) are available, they will be used to provide higher precision coordinates. If the RPC00B (or RPC00A) TRE is available, it is used to report RPC (Rapid Positioning Capability / Rational Polynomial Coefficients) metadata. RPC information can be retrieved from _rpc.txt files, and they will be used in priority over internal RPC00B values, since the latter have less precision than the ones stored in external _rpc.txt.

Most file header and image header fields are returned as dataset level metadata.

6.105.2 Driver capabilities

> **Supports CreateCopy()**
>
> This driver supports the *`GDALDriver::CreateCopy()`* operation

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.105.3 Open options

Added in version 3.7.

- **VALIDATE=[YES/NO]:** Defaults to `NO`. Whether TRE and DES (Data Extension Segment) content should be validated upon dataset opening. If errors are found, CE_Failure errors are emitted, but dataset opening does not fail, unless the FAIL_IF_VALIDATION_ERROR=YES open option is set. Note that validation is partial, and limited to the constraints documented in the nitf_spec.xml configuration file. Details of errors are also logged in `<error>` elements in the `xml:TRE` and `xml:DES` metadata domains.
- **FAIL_IF_VALIDATION_ERROR=[YES/NO]:** Defaults to `NO`. Whether validation errors reported by the *VALIDATE=YES* open option should prevent the dataset from being opened.

6.105.4 Creation Issues

On export NITF files are always written as NITF 2.1 with one image and no other auxiliary layers. Images are uncompressed by default, but JPEG and JPEG2000 compression are also available. Georeferencing can only be written for images using a geographic coordinate system or a UTM WGS84 projection. Coordinates are implicitly treated as WGS84 even if they are actually in a different geographic coordinate system. Pseudo-color tables may be written for 8bit images.

In addition to the export oriented CreateCopy() API, it is also possible to create a blank NITF file using Create() and write imagery on demand. However, using this methodology writing of pseudocolor tables and georeferencing is not supported unless appropriate IREP (Image Representation) and ICORDS (Image Coordinate) creation options are supplied.

Most file header, imagery header metadata and security fields can be set with appropriate **creation options** (although they are reported as metadata item, but must not be set as metadata). For instance setting *"FTITLE=Image of abandoned missile silo south west of Karsk"* in the creation option list would result in setting of the FTITLE field in the NITF file header. Use the official field names from the NITF specification document; do not put the "NITF_" prefix that is reported when asking the metadata list.

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following creation options are available:

- **IC=[NC/C3/C4/M3/C8]:** Defaults to `NC`. Set the compression method.
  - NC is the default value, and means no compression.
  - C3 means JPEG compression and is only available for the CreateCopy() method. The QUALITY and PROGRESSIVE JPEG-specific creation options can be used. See the *JPEG -- JPEG JFIF File Format*

driver. Multi-block images can be written.

  - M3 is a variation of C3. The only difference is that a block map is written, which allow for fast seeking to any block.
  - C4 means Vector Quantized (VQ) compression, and is only available when PRODUCT_TYPE=CADRG.
  - C8 means JPEG2000 compression (one block) and is available for CreateCopy() and/or Create() methods. See below paragraph for specificities.

- **QUALITY=value:** Defaults to `75`. For JPEG, quality as integer values in the 10-100 range For JPEG2000, quality as a floating-point value in >0 - 100 range. When JPEG2000_DRIVER=JP2OpenJPEG and PROFILE is not one of the NPJE ones, several quality layers can be specified as a comma-separated list of values.
- **PROGRESSIVE=[YES/NO]:** Defaults to `NO`. JPEG progressive mode
- **RESTART_INTERVAL=[-1/0/>0]:** Defaults to `-1`. Restart interval (in MCUs) for JPEG compression. -1 for auto, 0 for none, > 0 for user specified.
- **NUMI=value:** Defaults to `1`. Number of images. See *Writing multiple Image Segments* for the procedure to follow to write several images in a NITF file.
- **WRITE_ONLY_FIRST_IMAGE=[YES/NO]:** Defaults to `NO`. (Only taken into account if NUMI > 1, and on a new NITF file). If YES, only write first image. Subsequent ones must be written with APPEND_SUBDATASET=YES
- **WRITE_ALL_IMAGES=[YES/NO]:** (GDAL >= 3.4) Defaults to `NO`. (Only taken into account if NUMI > 1, and on a new NITF file). When set to NO (the default), this causes the driver to only write the first image segment and reserve just the space for extra NUMI-1 images in the file header. When WRITE_ALL_IMAGES=YES, the space for all images is allocated, which is only compatible with IC=NC (uncompressed images). (Behavior with GDAL < 3.4 was similar to WRITE_ALL_IMAGES=YES)
- **ICORDS=[G/D/N/S]:** Set to "G" to ensure that space will be reserved for geographic corner coordinates (in DMS) to be set later via SetGeoTransform(), set to "D" for geographic coordinates in decimal degrees, set to "N" for UTM WGS84 projection in Northern hemisphere or to "S" for UTM WGS84 projection in southern hemisphere (Only needed for Create() method, not CreateCopy()). If you Create() a new NITF file and have specified "N" or "S" for ICORDS, you need to call later the SetProjection method with a consistent UTM SRS to set the UTM zone number (otherwise it will default to zone 0). Starting with GDAL 3.5.1, when using the CreateCopy() interface with an image whose source SRS is a UTM WGS84 projection and specifying ICORDS=G or D, the NITF driver will reproject the image corner coordinates to longitude-latitude. This can be useful when it is not possible to encode in the IGEOLO field the coordinates of an image in the equatorial zone, whose one of the northing expressed in a UTM northern hemisphere projection is below -1e6.
- **IGEOLO=value:** (GDAL >= 3.5.1) Image corner coordinates specified as a string of 60 characters (cf MIL-STD-2500C for expected format). Normally automatically set from source geotransform and SRS when using the CreateCopy() interface. If specified, ICORDS must also be specified.
- **FHDR=[NITF02.10/NSIF01.00/NITF02.00]:** Defaults to `NITF02.10`. File version can be selected. "NITF02.10" (the default), and "NSIF01.00" are fully supported. Support for NITF02.00 has been introduced in GDAL 3.13. Note that for NITF02.00 writing UTM coordinates is not supported.
- **IREP=value:** Set to "RGB/LUT" (Look Up Table) to reserve space for a color table for each output band. (Only needed for Create() method, not CreateCopy()).
- **IREPBAND=value:** Comma separated list of band IREPBANDs in band order.
- **ISUBCAT=value:** Comma separated list of band ISUBCATs in band order.
- **LUT_SIZE=value:** Defaults to `256`. Set to control the size of pseudocolor tables for RGB/LUT bands. (Only needed for Create() method, not CreateCopy()).
- **BLOCKXSIZE=value:** Set the block width.

- **BLOCKYSIZE=value:** Set the block height.
- **BLOCKSIZE=value:** Set the block with and height. Overridden by BLOCKXSIZE and BLOCKYSIZE
- **BLOCKA_*=value:** If a complete set of BLOCKA options is provided with exactly the same organization as the NITF_BLOCKA metadata reported when reading an NITF file with BLOCKA TREs then a file will be created with BLOCKA TREs.
- **TRE=<tre-name=tre-contents>:** One or more TRE (Tagged Record Extension) creation options may be used provided to write arbitrary user defined TREs to the image header. The tre-name should be at most six characters, and the tre-contents should be "backslash escaped" if it contains backslashes or zero bytes. The argument is the same format as returned in the TRE metadata domain when reading.
- **FILE_TRE=<tre-name=tre-contents>:** Similar to above options, except that the TREs are written in the file header, instead of the image header.
- **RESERVE_SPACE_FOR_TRE_OVERFLOW=[YES/NO]:** (GDAL >= 3.6) Set to true to reserve space for IXSOFL when writing a TRE_OVERFLOW DES.
- **DES=<des-name=des-contents>:** One or more DES (Data Extension Segment) creation options may be provided to write arbitrary user defined DESs to the NITF file. The des-name should be at most 25 characters, and the des-contents should be "backslash escaped" if it contains backslashes or zero bytes, as in CPLEscapeString(str, -1, CPLES_BackslashQuotable). The des-contents must contain standard DES fields, starting with DESVER (See MIL-STD-2500C). DESs are not currently copied in CreateCopy(), but may be explicitly added as with Create().
- **NUMDES=value:** (GDAL >= 3.4) Number of DES segments. Only to be used on first image segment
- **TEXT=value:** TEXT options as text-option-name=text-option-content. Cf *Text Segments*.
- **CGM=value:** CGM options in cgm-option-name=cgm-option-content. Cf *CGM Segments*.
- **SDE_TRE=[YES/NO]:** Write GEOLOB (Local Geographic (lat/long) Coordinate System) and GEOPSB (Geo positioning Information) TREs to get more precise georeferencing. This is limited to geographic SRS, and to CreateCopy() for now.
- **RPC00B=[YES/NO]:** Defaults to `YES`. Write RPC00B TRE, from a source RPC00B TRE if it exists (NITF to NITF conversion), or from values found in the RPC metadata domain. This is only taken into account by CreateCopy() for now. Note that the NITF RPC00B format uses limited prevision ASCII encoded numbers.
- **RPCTXT=[YES/NO]:** Defaults to `NO`. Whether to write RPC metadata in a external _rpc.txt file. This may be useful since internal RPC00B TRE have limited precision. This is only taken into account by CreateCopy() for now.
- **USE_SRC_NITF_METADATA=[YES/NO]:** Defaults to `YES`. Whether to use NITF_xxx metadata items and TRE segments from the input dataset. It may needed to set this option to NO if changing the georeferencing of the input file.
- **PRODUCT_TYPE=[REGULAR/CADRG]:** (GDAL >= 3.13) Defaults to `REGULAR`. Sub-specification the output dataset should respect

The following options are only valid for PRODUCT_TYPE=CADRG.

- **COLOR_QUANTIZATION_BITS=[5/6/7/8]:** (GDAL >= 3.13) Defaults to `5`. Number of bits per R,G,B color component used during color palette computation. The higher the better quality and slower computation time. Only used when PRODUCT_TYPE=CADRG.
- **COLOR_TABLE_PER_FRAME=[YES/NO]:** (GDAL >= 3.13) Defaults to `NO`. Whether the color table should be optimized on the whole input dataset, or per output frame. The default is NO, that is optimized on the whole input dataset, to reduce the risk of color seams across frames. Only used when PRODUCT_TYPE=CADRG.
- **SCALE=value:** (GDAL >= 3.13) Reciprocal scale to use when generating output frames. Valid values are in the range from 1000 (1:1K) to 20000000 (1:20M). Special value `GUESS` can be also used to infer the scale

from the DPI, either explicitly specified with the DPI creation option, or if the TIFFTAG_YRESOLUTION / TIFFTAG_RESOLUTIONUNIT metadata items exist on the source raster. When not specified, the scale is inferred from the SERIES_CODE value from data series that have a fixed scale. Otherwise it is required. Only used when PRODUCT_TYPE=CADRG.

- **DPI=<float>:** (GDAL >= 3.13) Dot-Per-Inch value for the input dataset, that may need to be specified together with SCALE=GUESS. Valid values are in the range from 1 to 7200. If SCALE is not specified to the GUESS value, DPI is ignored. Only used when PRODUCT_TYPE=CADRG.
- **ZONE=<string>:** (GDAL >= 3.13) ARC Zone to which restrict generation of CADRG frames (1 to 9, A to H, J). If not specified, the driver automatically determines which zones the extent of the source dataset intersects. Only used when PRODUCT_TYPE=CADRG.
- **SERIES_CODE=[GN/JN/ON/TP/LF/JG/JA/JR/TF/AT/TC/TL/HA/CO/OA/CG/CM/MM]:** (GDAL >= 3.13) Defaults to `MM`. Two-letter code specifying the map/chart type. Used to infer the scale, when not specified, for some of the values where the map/chart type has a single nominal scale, and for the first 2 letters of the extension of frame files. Only used when PRODUCT_TYPE=CADRG. Below the codes from the initial version of MIL-STD-2411-1. Full table available in MIL-STD-2411/1 (W/CHANGE 3), pages 9-13.

Code	Data series	Scale
GN	Global Navigation Chart (GNC)	1:5 million
JN	Jet Navigation Chart (JNC)	1:2 million
ON	Operational Navigation Chart (ONC)	1:1 million
TP	Tactical Pilotage Chart (TPC)	1:500 K
LF	Low Flying Chart (LFC) - UK	1:500 K
JG	Joint Operation Graphic (JOG)	1:250 K
JA	Joint Operation Graphic, Air (JOG-A)	1:250 K
JR	Joint Operation Graphic, Radar (JOG-R)	1:250 K
TF	Transit Flying Chart (TFC) - UK	1:250 K
AT	Series 200 Air Target Chart (ATC)	1:200 K
TC	Topographic Line Map 100 (TLM 100)	1:100 K
TL	Topographic Line Map 50 (TLM 50)	1:50 K
HA	Harbor and Approach Charts (HAC)	Various
CO	Coastal Charts	Various
OP	Naval Range Operating Area Chart	Various
CG	City Graphics	Various
CM	Combat Charts	Various
MM	Miscellaneous Maps and Charts	Various

- **VERSION_NUMBER=<string>:** (GDAL >= 3.13) Defaults to `01`. Two letter version number (using letters among 0-9, A-H and J). Used for the 6th and 7th letters of the file name of frame files. Only used when PRODUCT_TYPE=CADRG.
- **PRODUCER_CODE_ID=<string>:** (GDAL >= 3.13) Defaults to `0`. One letter code indicating the data producer. This is a base-34 encoded value with valid letters '0' to '9', 'A' to 'Z' (excluding 'I' and 'O'). Used for the last letter of the file name of frame files. Only used when PRODUCT_TYPE=CADRG.

ID	Producer code	Producer
1	AFACC	Air Force Air Combat Command
2	AFESC	Air Force Electronic Systems Center
3	NIMA	National Imagery and Mapping Agency, Primary
4	NIMA1	NIMA, Alternate Site 1
5	NIMA2	NIMA, Alternate Site 2
6	NIMA3	NIMA, Alternate Site 3
7	SOCAF	Air Force Special Operations Command
8	SOCOM	United States Special Operations Command
9	PACAF	Pacific Air Forces
A	USAFE	United States Air Force, Europe
B	Non-DoD (NonDD)	US producer outside the Department of Defense
C	Non-US (NonUS)	Non-US producer
D	NIMA DCHUM (DCHUM)	NIMA produced Digital CHUM file
E	Non-NIMA DCHUM (DCHMD)	DoD producer of Digital CHUM file other than NIMA
F	Non-US DCHUM (DCHMF)	Non-US (foreign) producer of Digital CHUM files
G	Non-DoD DCHUM (DCHMG)	US producer of Digital CHUM files outside DoD
H	IMG2RPF	Non-specified, Imagery formatted to RPF
I–Z	Reserved	Reserved for future standardization

- **SECURITY_COUNTRY_CODE=<string>:** (GDAL >= 3.13) Two letter country ISO code of the security classification. Only used when PRODUCT_TYPE=CADRG.
- **CURRENCY_DATE=<string>:** (GDAL >= 3.13) Date of the most recent revision to the RPF product, as YYYYMMDD. Can be set to empty to avoid writing it, or the special value NOW for the current date, otherwise a default value of 20260101 is used. Only used when PRODUCT_TYPE=CADRG.
- **PRODUCTION_DATE=<string>:** (GDAL >= 3.13) Date that the source data was transferred to RPF format, as YYYYMMDD. Can be set to empty to avoid writing it, or the special value NOW for the current date, otherwise a default value of 20260101 is used. Only used when PRODUCT_TYPE=CADRG.
- **SIGNIFICANT_DATE=<string>:** (GDAL >= 3.13) Date describing the basic date of the source product, as YYYYMMDD. Can be set to empty to avoid writing it, or the special value NOW for the current date, otherwise a default value of 20260101 is used. Only used when PRODUCT_TYPE=CADRG.
- **DATA_SERIES_DESIGNATION=<string>:** (GDAL >= 3.13) Short title for the identification of a group of products usually having the same scale and/or cartographic specification (e.g. JOG 1501A). Up to 10 characters. Derived from SERIES_CODE and SCALE when not specified. Only used when PRODUCT_TYPE=CADRG.
- **MAP_DESIGNATION=<string>:** (GDAL >= 3.13) Designation, within the data series, of the hard-copy source (e.g. G18 if the hard-copy source is ONC G18). Up to 8 characters. Only used when PRODUCT_TYPE=CADRG.

The following creation options to set fields in the NITF file header are available:

- **OSTAID=string of up to 10 characters:** Originating Station ID
- **FDT=string of up to 14 characters:** File Date and Time. Format is DDHHMMSSZMONYY for NITF 2.0 and CCYYMMDDhhmmss for NITF 2.1. `NOW` is also accepted as a special value for the current date-time.
- **FTITLE=string of up to 80 characters:** File Title
- **FSCLAS=string of 1 character:** File Security Classification (U/R/C/S/T)
- **FSCLSY=string of up to 2 characters:** File Classification Security System (NITF02.10/NSIF only)
- **FSCODE=string of up to 11 characters:** File Codewords (NITF02.10/NSIF only)
- **FSCTLH=string of up to 2 characters:** File Control and Handling

- **FSREL=string of up to 20 characters:** File Releasing Instructions
- **FSDCTP=string of up to 2 characters:** File Declassification Type (NITF02.10/NSIF only)
- **FSDCDT=string of 8 characters:** File Declassification Date (NITF02.10/NSIF only)
- **FSDCXM=string of up to 4 characters:** File Declassification Exemption (NITF02.10/NSIF only)
- **FSDG=string of 1 character:** File Downgrade (NITF02.10/NSIF only)
- **FSDGDT=string of 8 characters:** File Downgrade Date (NITF02.10/NSIF only)
- **FSDWNG=string of up to 6 characters:** File Security Downgrade (NITF02.00 only)
- **FSDEVT=string of 40 characters:** File Downgrading Event (NITF02.00 only)
- **FSCLTX=string of up to 43 characters:** File Classification Text (NITF02.10/NSIF only)
- **FSCATP=string of 1 character:** File Classification Authority Type (NITF02.10/NSIF only)
- **FSCAUT=string of up to 40 characters:** File Classification Authority
- **FSCRSN=string of 1 character:** File Classification Reason
- **FSSRDT=string of 8 characters:** File Security Source Date (NITF02.10/NSIF only)
- **FSCTLN=string of up to 15 characters:** File Security Control Number (NITF02.10/NSIF only)
- **FSCOP=string of up to 5 characters:** File Copy Number
- **FSCPYS=string of up to 5 characters:** File Number of Copies
- **ONAME=string of up to 24 characters:** Originator Name
- **OPHONE=string of up to 18 characters:** Originator Phone Number

The following creation options to set fields in the NITF image header are available for NITF 02.10 and NSIF 1.0. For NITF 02.00, consult MIL-STD-2500A.

- **IID1=string of up to 10 characters:** Image Identifier 1
- **IDATIM=string of 14 characters:** Image Date and Time. Format is DDHHMMSSZMONYY for NITF 2.0 and CCYYMMDDhhmmss for NITF 2.1. `NOW` is also accepted as a special value for the current date-time.
- **TGTID=string of up to 17 characters:** Target Identifier
- **IID2=string of up to 80 characters:** Image Identifier 2 (NITF02.10/NSIF only)
- **ITITLE=string of up to 80 characters:** Image Title (NITF02.00 only)
- **ISCLAS=string of 1 character:** Image Security Classification (U/R/C/S/T)
- **ISCLSY=string of up to 2 characters:** Image Classification Security System (NITF02.10/NSIF only)
- **ISCODE=string of up to 11 characters:** Image Codewords
- **ISCTLH=string of up to 2 characters:** Image Control and Handling
- **ISREL=string of up to 20 characters:** Image Releasing Instructions (NITF02.10/NSIF only)
- **ISDCTP=string of up to 2 characters:** Image Declassification Type (NITF02.10/NSIF only)
- **ISDCDT=string of 8 characters:** Image Declassification Date (NITF02.10/NSIF only)
- **ISDCXM=string of up to 4 characters:** Image Declassification Exemption (NITF02.10/NSIF only)
- **ISDG=string of 1 character:** Image Downgrade (NITF02.10/NSIF only)
- **ISDGDT=string of 8 characters:** Image Downgrade Date (NITF02.10/NSIF only)
- **ISDWNG=string of up to 6 characters:** Image Security Downgrade (NITF02.00 only)

- **ISDEVT=string of 40 characters:** Image Downgrading Event (NITF02.00 only)
- **ISCLTX=string of up to 43 characters:** Image Classification Text (NITF02.10/NSIF only)
- **ISCATP=string of 1 character:** Image Classification Authority Type (NITF02.10/NSIF only)
- **ISCAUT=string of up to 40 characters:** Image Classification Authority
- **ISCRSN=string of 1 character:** Image Classification Reason (NITF02.10/NSIF only)
- **ISSRDT=string of 8 characters:** Image Security Source Date (NITF02.10/NSIF only)
- **ISCTLN=string of up to 15 characters:** Image Security Control Number (NITF02.10/NSIF only)
- **ISORCE=string of up to 42 characters:** Image Source
- **ICAT=string of up to 8 characters:** Image Category
- **ABPP=integer:** Actual Bits-Per-Pixel Per Band. Starting with GDAL 3.10, also available as the `NBITS` creation option.
- **PJUST=string of 1 character:** Pixel Justification
- **ICOM=string of up to 720 characters:** Image Comments (organized as up to 9 lines of 80 characters)
- **IDLVL=integer of up to 3 characters:** Image Display Level
- **IALVL=integer of up to 3 characters:** Image Attachment Level
- **ILOCROW=integer of up to 5 characters:** Image Location Row
- **ILOCCOL=integer of up to 5 characters:** Image Location Column

6.105.5 JPEG2000 compression (write support)

JPEG2000 compression is available when using the IC=C8 creation option, if the JP2ECW (SDK 3.3, or for later versions assuming the user has the key to enable JPEG2000 writing), JP2KAK or JP2OpenJPEG driver are available.

They are tried in that order when several ones are available, unless the JPEG2000_DRIVER creation option (added in GDAL 3.4) is set to explicitly specify the JPEG2000 capable driver to use.

- *JP2ECW*: The *TARGET* (target size reduction as a percentage of the original) and *PROFILE*=BASELINE_0/BASELINE_1/BASELINE_2/NPJE/EPJE JP2ECW-specific creation options can be used. Both CreateCopy() and/or Create() methods are available. By default the NPJE PROFILE will be used (thus implying BLOCKXSIZE=BLOCKYSIZE=1024).
- *JP2KAK*: The *QUALITY*, *BLOCKXSIZE*, *BLOCKYSIZE*, *LAYERS*, *ROI* JP2KAK-specific creation options can be used. Only CreateCopy() method is available.
- *JP2OpenJPEG*: (only in the CreateCopy() case). The *QUALITY*, *BLOCKXSIZE* and *BLOCKYSIZE*, *TARGET* (target size reduction as a percentage of the original) JP2OpenJPEG-specific creation options can be used. By default BLOCKXSIZE=BLOCKYSIZE=1024 will be used.

  Starting with GDAL 3.4.0 and OpenJPEG 2.5, the PROFILE=NPJE_VISUALLY_LOSSLESS/NPJE_NUMERICALLY_LOSSLES creation option can be used to create files that comply with STDI-0006 NITF Version 2.1 Commercial Dataset Requirements Document (NCDRD). For NPJE_VISUALLY_LOSSLESS, the last quality layer defaults to 3.9 bits per pixel and per band. It can be adjusted with the QUALITY creation option. When those profiles are specified, the J2KLRA TRE will also be written, unless the `J2KLRA=NO` creation option is specified.

6.105.6 CADRG (Compressed ARC Digitized Raster Graphics) (write support)

Added in version 3.13.

The driver supports generating CADRG frames from any georeferenced source dataset, which has a single band with a color table of up to 216 colors, a 3-band RGB source dataset or a 4-band RGBA datasets. It will automatically create the CADRG frames intersecting the source dataset extent in the appropriate Arc zones, at the target scale. The *PRODUCT_TYPE* creation option must be set to `CADRG`. The output dataset name must be a directory name.

The target scale is determined by decreasing order of priority:

- from the value of the *SCALE* creation option, if specified.
- from the value of the *DPI* creation option, if specified.
- from the value of the *SERIES_CODE* creation option, if a scale is associated with it (which is the case of all valid codes except `HA`, `CO`, `OP`, `CG`, `CM` and `MM`)
- from the value of the TIFFTAG_RESOLUTIONUNIT and TIFFTAG_YRESOLUTION metadata items, if existing in the source dataset.

By default, hard-coded values of dates/date-times are written in the file, so as to get binary reproducible outputs from a given input. It is possible to customize them by setting the *FDT*, *IDATIM*, *CURRENCY_DATE*, *PRODUCTION_DATE* and *SIGNIFICANT_DATE* creation options. For all of them the special value `NOW` can be specified to use the current timestamp.

The frame index `A.TOC` file is automatically generated (it can also be generated manually with the *gdal driver rpftoc create* program).

6.105.6.1 Example 1: Create CADRG frames from a VRT mosaic, specifying it is an Operational Navigation Chart (scale 1:1 million), using hard-coded dates

```
gdal raster convert input_mosaic.vrt output_directory \
    --format=NITF --co PRODUCT_TYPE=CADRG --co SERIES_CODE=ON
```

6.105.6.2 Example 2: Create CADRG frames from a VRT mosaic, specifying it is a City Graphic map at scale 1:5,000, with dates corresponding to the current timestamp.

```
gdal raster convert input_mosaic.vrt output_directory \
    --format=NITF --co PRODUCT_TYPE=CADRG --co SERIES_CODE=CG --co SCALE=5000 \
    --co OSTAID=MyCompany --co ONAME=Norway --co PRODUCER_CODE_ID=X --co SECURITY_
↪COUNTRY_CODE=NO \
    --co FDT=NOW --co IDATIM=NOW --co CURRENCY_DATE=NOW \
    --co PRODUCTION_DATE=NOW --co SIGNIFICANT_DATE=NOW
```

6.105.7 Links

- *Advanced GDAL NITF Driver Information*
- NITFS Technical Board Public Page
- DIGEST Part 2 Annex D (describe encoding of NITF Spatial Data Extensions)
- *RPFTOC -- Raster Product Format/RPF (a.toc)*: to read the Table Of Contents of CIB and CADRG products.
- MIL-PRF-89038 : specification of RPF, CADRG, CIB products
- *ECRGTOC -- ECRG Table Of Contents (TOC.xml)*: to read the Table Of Contents of ECRG products.
- MIL-PRF-32283 : specification of ECRG products

6.105.8 Credit

The author wishes to thank AUG Signals and the GeoConnections program for supporting development of this driver, and to thank Steve Rawlinson (JPEG), Reiner Beck (BLOCKA) for assistance adding features.

6.106 NOAA_B -- NOAA GEOCON/NADCON5 .b format

Added in version 3.7.

Driver short name

NOAA_B

Driver built-in by default

This driver is built-in by default

GDAL supports reading geodetic grids in the NOAA .b binary format used for GEOCON and NADCON5 grids.

6.106.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.106.2 See also

- Format specification: See paragraph 10.2, ".b grids (GEOCON and NADCON 5.0)"

6.107 NSIDCbin -- National Snow and Ice Data Centre Sea Ice Concentrations

Driver short name

NSIDCbin

Driver built-in by default

This driver is built-in by default

Added in version 3.7.

Supported by GDAL for read access. This format is a raw binary format for the Nimbus-7 SMMR and DMSP SSM/I-SSMIS Passive Microwave Data sea ice concentrations. There are daily and monthly maps in the north and south hemispheres supported by this driver.

Support includes an affine georeferencing transform, and projection - these are both 25000m resolution polar stereographic grids centred on the north and south pole respectively. Metadata from the file including julian day and year are recorded.

This driver is implemented based on the NSIDC documentation in the User Guide.

Band values are Byte, sea ice concentration (fractional coverage scaled by 250).

The dataset band implements GetScale() which will convert the values from 0,255 to 0.0,102.0 by multiplying by 0.4. Unscaled values above 250 have specific meanings, 251 is Circular mask used in the Arctic, 252 is Unused, 253 is Coastlines, 254 is Superimposed land mask, 255 is Missing data.

NOTE: Implemented as nsidcbindataset.cpp.

6.107.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.107.2 Example

For example, we want to read monthly data from September 2019, with data from this source (requires authentication).

<https://n5eil01u.ecs.nsidc.org/PM/NSIDC-0051.001/2018.09.01/nt_201809_f17_v1.1_s.bin>

```
gdalinfo nt_201809_f17_v1.1_s.bin
```

6.108 NTv2 -- NTv2 Datum Grid Shift

Driver short name

NTv2

Driver built-in by default

This driver is built-in by default

NOTE: Implemented as ntv2dataset.cpp.

6.108.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.109 NWT_GRD/NWT_GRC -- Northwood/Vertical Mapper File Format

Driver short name

NWT_GRD

Driver short name

NWT_GRC

Driver built-in by default

This driver is built-in by default

Support for reading & writing Northwood GRID raster formats. This format is also known as Vertical Mapper Grid or MapInfo Grid and is commonly used in MapInfo Pro software

Full read/write support of *.grd (grid) files is available, read-only support is available for classified grids (*.grc).

For writing, Float32 is the only supported band type.

6.109.1 Driver capabilities (NWT_GRD)

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.109.2 Driver capabilities (NWT_GRC)

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.109.3 Color Information

The grid formats have color information embedded in the grid file header. This describes how to scale data values to RGB values. When opening in read mode, the driver will report 4 bands - R, G, B and the data band. In reality there is 1 band and the RGB bands are 'virtual', made from scaling data. For this reason, when opening in write mode only 1 band is reported and the RGB bands are unavailable.

6.109.4 Metadata

GDAL Metadata items are stored in the PAM .aux.xml file

Northwood Grid itself does not natively support arbitrary metadata

6.109.5 Nodata values

In write mode, it is possible to designate any value as the nodata value. These values are translated to the Vertical Mapper no data value when writing. Therefore, in read mode the nodata value is always reported as -1e37.

6.109.5.1 Creation Options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following creation options are available:

- **ZMIN=value:** Defaults to `-2e37`. Set the minimum Z value. Data are scaled on disk to a 16 bit integer and the Z value range is used to scale data. If not set, it may cause incorrect data to be written when using 'Create()' or a full recalculation of the source dataset statistics when using 'CreateCopy'
- **ZMAX=value:** Defaults to `2e38`. Set the maximum Z value. Data are scaled on disk to a 16 bit integer and the Z value range is used to scale data. If not set, it may cause incorrect data to be written when using 'Create()' or a full recalculation of the source dataset statistics when using 'CreateCopy'
- **BRIGHTNESS=value:** Defaults to `50`. Set the brightness level. Only affects opening the file in MapInfo/Vertical Mapper

- **CONTRAST=value:** Defaults to `50`. Set the contrast level. Only affects opening the file in MapInfo/Vertical Mapper
- **TRANSCOLOR=value:** Defaults to `0`. Set a transparent color level. Only affects opening the file in MapInfo/Vertical Mapper
- **TRANSLUCENCY=value:** Defaults to `0`. Set the translucency level. Only affects opening the file in MapInfo/Vertical Mapper

6.110 OGCAPI -- OGC API Tiles / Maps / Coverage

Added in version 3.2.

Driver short name

OGCAPI

Build dependencies

libcurl

Access to server implementing OGC API - Tiles, OGC API - Maps or OGC API - Coverages. This driver has raster and vector capabilities.

Warning

This driver is experimental, and has been developed to demonstrate work related to the "Modular OGC API Workflows" initiative. It implements non-finalized versions of OGC API - Tiles, - Maps and - Coverages. Its interface may change at any time, or it might be removed. It might also be eventually merged with the OGC API - Features driver.

6.110.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

6.110.2 Dataset opening

The driver supports opening by:

- passing a filename (with .moaw extension) containing a JSON document, like the following, specifying a deferred processing

```
{
    "process" : "https://maps.ecere.com/ogcapi/processes/RenderMap",
    "inputs" : {
        "transparent" : false,
        "background" : "navy",
        "layers" : [
```

(continues on next page)

(continued from previous page)

```
            { "collection" : "https://maps.ecere.com/ogcapi/collections/
↪NaturalEarth:physical:bathymetry" },
            { "collection" : "https://maps.ecere.com/ogcapi/collections/SRTM_
↪ViewFinderPanorama" }
        ]
    }
}
```

- passing a string "OGCAPI:{url}" where {url} is the URL to a OGC API landing page In that case the driver will return subdatasets with the different collections.

  Starting with GDAL 3.10, specifying the `-if OGCAPI` option to command line utilities accepting it, or `OGCAPI` as the only value of the `papszAllowedDrivers` of *GDALOpenEx()*, also forces the driver to recognize the passed URL, without the `OGCAPI:` prefix.

- passing a string "OGCAPI:{url}" where {url} is the URL to a OGC API collection description

When the driver opens a collection, for raster, it will look if tiles or maps API are advertized for it. It will use tiles API by default, and fallback to maps API when not available. It will also look at the image formats, and will prefer PNG When available.

For vector collections, this driver handles the tiles API, with GeoJSON or Mapbox Vector tiles.

When using the tiles API, the driver will use by default the WorldCRS84Quad tile matrix set when available

6.110.3 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are available:

- **API=[AUTO/MAPS/TILES/COVERAGE/ITEMS]:** Defaults to `AUTO`. Which API to use for data acquisition. In AUTO mode, for raster access, coverage is used if available, and fallback first to tiles and finally maps otherwise. In AUTO mode, for vector access, tiles is used if available, and fallback to GeoJSON items otherwise.
- **IMAGE_FORMAT=[AUTO/PNG/PNG_PREFERRED/JPEG/JPEG_PREFERRED/GEOTIFF]:** Defaults to `AUTO`. Which format to use for pixel acquisition, for tiles or map API. AUTO - This is the default and specifies that PNG images will be checked first, then JPEG and then any additional formats the server supports. PNG_PREFERRED - Same as AUTO JPEG_PREFERRED - Similar to AUTO, but the order is JPEG, PNG and then any additional formats the server supports JPEG - Use only JPEG images. If none are available then the driver will return an error PNG - Use only PNG images. If none are available then the driver will return an error GEOTIFF - Use only GEOTIFF images. If none are available then the driver will return an error
- **VECTOR_FORMAT=[AUTO/GEOJSON/GEOJSON_PREFERRED/MVT/MVT_PREFERRED]:** Defaults to `AUTO`. Which format to use for vector data acquisition. Defaults to AUTO, which means that MVT (Mapbox Vector Tiles) will be used if available, and fallback to GEOJSON otherwise. If specifying MVT or GEOJSON, they must be available, otherwise the driver will return an error. If specifying the one of the MVT_PREFERRED or GEOJSON_PREFERRED value, the specified format will be used if available, and the driver will fallback to the other format otherwise.
- **TILEMATRIXSET=<id>:** Identifier of the required tile matrix set. Only used with the tiles API. If this tile matrix set is not available, the driver will fail. If this option is not specified, the driver will automatically select one of the available tile matrix sets. TILEMATRIXSET and PREFERRED_TILEMATRIXSET options are mutually exclusive.
- **PREFERRED_TILEMATRIXSET=<id>:** Identifier of the preferred tile matrix set. Only used with the tiles API. If this tile matrix set is not available, or if this option is not specified, the driver will automatically select one of the available tile matrix sets. TILEMATRIXSET and PREFERRED_TILEMATRIXSET options are mutually exclusive.

- **TILEMATRIX=<id>:** Identifier of a particular tile matrix (zoom level) of the select tile matrix set. If not specified, all available tile matrix are returned as overviews (for raster data), or layers (for vector data)
- **CACHE=[YES/NO]:** Defaults to YES. Whether to enable block/tile caching. Only for tiles API, and with raster data.
- **MAX_CONNECTIONS=<int>:** Defaults to 5. Maximum number of connections for parallel tile downloading. Only for tiles API, and with raster data.
- **MINX=value:**
- **MINY=value:**
- **MAXX=value:**
- **MAXY=value:** Bounds in SRS of TileMatrixSet to which to restrict the exposed dataset/layers.

6.111 ESRI File Geodatabase raster (OpenFileGDB)

Added in version 3.7.

Driver short name

OpenFileGDB

Driver built-in by default

This driver is built-in by default

The OpenFileGDB driver provides read access to raster layers of File Geodatabases (.gdb directories). The dataset name must be the directory/folder name, and it must end with the .gdb extension.

It can also read directly zipped .gdb directories (with .gdb.zip extension), provided they contain a .gdb directory at their first level.

The driver supports:

- reading CRS information
- reading geotransform
- exposing overviews
- exposing nodata mask band or nodata value
- uncompressed, LZ77, JPEG and JPEG2000 compression methods.
- exposing value attribute tables as GDAL Raster attribute tables.

Support for FileGDB created by ArcGIS v10 has been added in GDAL 3.7 Support for FileGDB created by ArcGIS v9 has been added in GDAL 3.8

6.111.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.111.2 Connection string

When a File Geodatabase contains several raster layers, the raster layer that must be opened is specified with the following connection string

```
OpenFileGDB:"/path/to/my.gdb":name_of_raster_layer
```

When opening a dataset with pointing only to the .gdb directory, and if it contains several raster layers, a list of subdatasets is returned by the driver

6.111.3 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are available:

- **NODATA_OR_MASK=[AUTO/MASK/NONE/numeric nodata value]:** Control nodata handling.
  - In AUTO mode, the driver will expose a dataset nodata mask band, unless the band data type is Float32 or Float64, in which case a nodata value is used.
  - In MASK mode, the driver will expose a dataset nodata mask band for all data types.
  - In NONE mode, the driver will not expose a nodata mask band or a nodata value.
  - When specifying a numeric nodata value (`nan` accepted for Float32 or Float64), it is used as the band nodata value. The nodata value should be selected outside the range of valid values (but within the range of the data type).

6.111.4 Metadata

The `xml:documentation` and `xml:definition` metadata domains contain the XML content from the `GDB_Items` table related to the raster layer.

6.111.5 Examples

- List raster layers from a FileGDB

```
gdalinfo /path/to/my.gdb
```

- Open a given subdataset:

```
gdalinfo OpenFileGDB:"/path/to/my.gdb":name_of_raster_layer
```

6.111.6 Links

- *OpenFileGDB vector* documentation page

6.111.7 Credits

Thanks to Richard Barnes and his ArcRescue tool for the deciphering of the band_types field which indicates the compression method and the data type.

6.112 JAXA PALSAR Processed Products

Driver short name

JAXAPALSAR

Driver built-in by default

This driver is built-in by default

This driver provides enhanced support for processed PALSAR products from the JAXA PALSAR processor. This encompasses products acquired from the following organizations:

- JAXA (Japanese Aerospace eXploration Agency)
- AADN (Alaska Satellite Facility)
- ESA (European Space Agency)

This driver does not support products created using the Vexcel processor (i.e. products distributed by ERSDAC and affiliated organizations).

Support is provided for the following features of PALSAR products:

- Reading Level 1.1 and 1.5 processed products
- Georeferencing for Level 1.5 products
- Basic metadata (sensor information, ground pixel spacing, etc.)
- Multi-channel data (i.e. dual-polarization or fully polarimetric datasets)

This is a read-only driver.

To open a PALSAR product, select the volume directory file (for example, VOL-ALPSR000000000-P1.5_UA or VOL-ALPSR000000000-P1.1__A). The driver will then use the information contained in the volume directory file to find the various image files (the IMG-* files), as well as the Leader file. Note that the Leader file is essential for correct operation of the driver.

6.112.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.112.2 See Also

- RESTEC Sample Data

6.113 PAux -- PCI .aux Labelled Raw Format

Driver short name

PAux

Driver built-in by default

This driver is built-in by default

GDAL includes a partial implementation of the PCI .aux labelled raw raster file for read, write and creation. To open a PCI labelled file, select the raw data file itself. The .aux file (which must have a common base name) will be checked for automatically.

The format type for creating new files is `PAux`. All PCI data types (8U, 16U, 16S, and 32R) are supported. Currently georeferencing, projections, and other metadata is ignored.

See Also: PCI's .aux Format Description

6.113.1 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.114 PCIDSK -- PCI Geomatics Database File

Driver short name

PCIDSK

Driver built-in by default

This driver is built-in by default

PCIDSK database file used by PCI EASI/PACE software for image analysis. It is supported for reading, and writing by GDAL. All pixel data types, and data organizations (pixel interleaved, band interleaved, file interleaved and tiled) should be supported. Currently LUT segments are ignored, but PCT segments should be treated as associated with the bands. Overall file, and band specific metadata should be correctly associated with the image or bands.

Georeferencing is supported though there may be some limitations in support of datums and ellipsoids. GCP segments are ignored. RPC segments will be returned as GDAL style RPC metadata.

Internal overview (pyramid) images will also be correctly read though newly requested overviews will be built externally as an .ovr file.

Vector segments are also supported by the driver.

6.114.1 Creation Options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following creation options are supported:

Note that PCIDSK files are always produced pixel interleaved, even though other organizations are supported for read.

- **INTERLEAVING=[PIXEL/BAND/FILE/TILED]:** sets the interleaving for the file raster data.
- **COMPRESSION=[NONE/RLE/JPEG]:** Defaults to `NONE`. Sets the compression to use. Values other than NONE may only be used with TILED interleaving. If JPEG is select it may include a quality value between 1 and 100 - eg. COMPRESSION=JPEG40.
- **TILESIZE=value:** Defaults to `127`. When INTERLEAVING is TILED, the tilesize may be selected with this parameter - the default is 127 for 127x127.

6.114.2 Driver capabilities

Supports CreateCopy()

This driver supports the *`GDALDriver::CreateCopy()`* operation

Supports Create()

This driver supports the *`GDALDriver::Create()`* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.114.3 See Also:

- Implemented as pcidskdataset2.cpp.
- PCIDSK SDK

6.115 PCRaster -- PCRaster raster file format

Driver short name

PCRaster

Build dependencies

(internal libcf provided)

GDAL includes support for reading and writing PCRaster raster files. PCRaster is a dynamic modeling system for distributed simulation models. The main applications of PCRaster are found in environmental modeling: geography, hydrology, ecology to name a few. Examples include models for research on global hydrology, vegetation competition models, slope stability models and land use change models.

The driver reads all types of PCRaster maps: booleans, nominal, ordinals, scalar, directional and ldd. The same cell representation used to store values in the file is used to store the values in memory.

The driver detects whether the source of the GDAL raster is a PCRaster file. When such a raster is written to a file the value scale of the original raster will be used. The driver **always** writes values using UINT1, INT4 or REAL4 cell representations, depending on the value scale:

Value scale	Cell representation
VS_BOOLEAN	CR_UINT1
VS_NOMINAL	CR_INT4
VS_ORDINAL	CR_INT4
VS_SCALAR	CR_REAL4
VS_DIRECTION	CR_REAL4
VS_LDD	CR_UINT1

For rasters from other sources than a PCRaster raster file a value scale and cell representation is determined according to the following rules:

Source type	Target value scale	Target cell representation
GDT_UInt8	VS_BOOLEAN	CR_UINT1
GDT_Int32	VS_NOMINAL	CR_INT4
GDT_Float32	VS_SCALAR	CR_REAL4
GDT_Float64	VS_SCALAR	CR_REAL4

The driver can convert values from one supported cell representation to another. It cannot convert to unsupported cell representations. For example, it is not possible to write a PCRaster raster file from values which are used as CR_INT2 (GDT_Int16).

Although the de-facto file extension of a PCRaster raster file is .map, the PCRaster software does not require a standardized file extension.

NOTE: Implemented as pcrasterdataset.cpp.

See also: PCRaster website at Utrecht University.

6.115.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.116 PDF -- Geospatial PDF

Driver short name

PDF

Build dependencies

none for write support, Poppler/PoDoFo/PDFium for read support

GDAL supports reading Geospatial PDF documents, by extracting georeferencing information and rasterizing the data. Non-geospatial PDF documents will also be recognized by the driver.

PDF documents can be created from other GDAL raster datasets, and OGR datasources can also optionally be drawn on top of the raster layer (see OGR_* creation options in the below section).

The driver supports reading georeferencing encoded in either of the 2 current existing ways : according to the Adobe Supplement to ISO 32000, or to a so-called OGC encoding "best practice" (which is not recommended at all).

Multipage documents are exposed as subdatasets, one subdataset par page of the document.

6.116.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.116.2 Vector support

See the *PDF vector* documentation page

6.116.3 Metadata

The neatline (for OGC best practice) or the bounding box (Adobe style) will be reported as a NEATLINE metadata item, so that it can be later used as a cutline for the warping algorithm.

XMP metadata can be extracted from the file, and will be stored as XML raw content in the xml:XMP metadata domain.

Additional metadata, such as found in USGS Topo PDF can be extracted from the file, and will be stored as XML raw content in the EMBEDDED_METADATA metadata domain.

6.116.4 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The following configuration options are supported:

- **GDAL_PDF_DPI=value:** Defaults to `150`. To control the dimensions of the raster by specifying the DPI of the rasterization. The driver will make some effort to guess the DPI value either from a specific metadata item contained in some PDF files, or from the raster images inside the PDF (in simple cases).
- **GDAL_PDF_NEATLINE=value:** The name of the neatline to select (only available for geospatial PDF, encoded according to OGC Best Practice). This defaults to "Map Layers" for USGS Topo PDF. If not found, the neatline that covers the largest area.
- **PDF_USER_PWD=value:** User password for protected PDFs.
- **GDAL_PDF_RENDERING_OPTIONS=[RASTER,VECTOR,TEXT/RASTER,VECTOR/RASTER,TEXT/RASTER/VECTOR,TEXT/VECTOR/TEXT]:** a combination of VECTOR, RASTER and TEXT separated by comma, to select whether vector, raster or text features should be rendered. If the option is not specified, all features are rendered (Poppler and PDFium).
- **GDAL_PDF_BANDS=[3/4]:** whether the PDF should be rendered as a RGB (3) or RGBA (4) image. The default value will depend on the PDF rendering used (Poppler vs PDFium) and on the content found in the PDF file (if an image with transparency is recognized, then 4 will be used). When 3 bands is selected, a white background is used.
- **GDAL_PDF_LAYERS=value:** list of layers (comma separated) to turn ON (or "ALL" to turn all layers ON). The layer names can be obtained by querying the LAYERS metadata domain. When this option is specified, layers not explicitly listed will be turned off (Poppler and PDFium).
- **GDAL_PDF_LAYERS_OFF=value:** list of layers (comma separated) to turn OFF. The layer names can be obtained by querying the LAYERS metadata domain (Poppler and PDFium).
- **GDAL_PDF_LAUNDER_LAYER_NAMES=[YES/NO]:** (GDAL >= 3.1) Can be set to NO to avoid the layer names reported in the LAYERS metadata domain or as OGR layers for the vector part to be "laundered".

6.116.4.1 Open Options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). Above configuration options are also available as open options.

- **RENDERING_OPTIONS=[RASTER,VECTOR,TEXT/RASTER,VECTOR/RASTER,TEXT/RASTER/VECTOR,TEXT/VECTOR/TEXT]:** Equivalent of *GDAL_PDF_RENDERING_OPTIONS* configuration option.
- **DPI=value:** Equivalent of *GDAL_PDF_DPI* configuration option
- **SAVE_DPI_TO_PAM=[YES/NO]:** (GDAL >= 3.13) Defaults to `NO`. Whether to save the DPI value provided with the *DPI* open option into PAM (.aux.xml) metadata as the new default DPI that will be used when re-opening the dataset.
- **USER_PWD=value:** Equivalent of *PDF_USER_PWD* configuration option
- **PDF_LIB=[POPPLER/PODOFO/PDFIUM]:** only available for builds with multiple backends.
- **LAYERS=value:** Equivalent of *GDAL_PDF_LAYERS* configuration option
- **LAYERS_OFF=value:** Equivalent of *GDAL_PDF_LAYERS_OFF* configuration option
- **BANDS=value:** Equivalent of *GDAL_PDF_BANDS* configuration option
- **NEATLINE=value:** Equivalent of as *GDAL_PDF_NEATLINE* configuration option

6.116.5 LAYERS Metadata domain

When GDAL is compiled against Poppler or PDFium, the LAYERS metadata domain can be queried to retrieve layer names that can be turned ON or OFF. This is useful to know which values to specify for the *GDAL_PDF_LAYERS* or *GDAL_PDF_LAYERS_OFF* configuration options.

For example :

```
$ gdalinfo ../autotest/gdrivers/data/adobe_style_geospatial.pdf -mdd LAYERS

Driver: PDF/Geospatial PDF
Files: ../autotest/gdrivers/data/adobe_style_geospatial.pdf
[...]
Metadata (LAYERS):
  LAYER_00_NAME=New_Data_Frame
  LAYER_01_NAME=New_Data_Frame.Graticule
  LAYER_02_NAME=Layers
  LAYER_03_NAME=Layers.Measured_Grid
  LAYER_04_NAME=Layers.Graticule
[...]

$ gdal_translate ../autotest/gdrivers/data/adobe_style_geospatial.pdf out.tif --config␣
↪GDAL_PDF_LAYERS_OFF "New_Data_Frame"
```

6.116.6 Restrictions

The opening of a PDF document (to get the georeferencing) is fast, but at the first access to a raster block, the whole page will be rasterized (with Poppler), which can be a slow operation.

Note: some raster-only PDF files (such as some USGS PDF files), that are regularly tiled are exposed as tiled dataset by the GDAL PDF driver, and can be rendered with any backends.

Only a few of the possible Datums available in the OGC best practice spec have been currently mapped in the driver. Unrecognized datums will be considered as being based on the WGS84 ellipsoid.

For documents that contain several neatlines in a page (insets), the georeferencing will be extracted from the inset that has the largest area (in term of screen points).

6.116.7 Creation Issues

PDF documents can be created from other GDAL raster datasets, that have 1 band (graylevel or with color table), 3 bands (RGB) or 4 bands (RGBA).

Georeferencing information will be written by default according to the ISO32000 specification.

Note: PDF write support does not require linking to any backend.

6.116.7.1 Creation Options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following creation options are available:

- **COMPRESS=[NONE/DEFLATE/JPEG/JPEG2000]:** Defaults to `DEFLATE`. Set the compression to use for raster data.
- **STREAM_COMPRESS=[NONE/DEFLATE]:** Defaults to `DEFLATE`. Set the compression to use for stream objects (vector geometries, JavaScript content).
- **DPI=value:** Defaults to `72`. Set the DPI to use. May be automatically adjusted to higher value so that page dimension does not exceed the 14400 maximum value (in user units) allowed by Acrobat.
- **WRITE_USERUNIT=[YES/NO]:** Whether the UserUnit setting computed from the DPI (UserUnit = DPI / 72.0) should be recorded in the file. When UserUnit is recorded, the raster size in pixels recognized by GDAL on reading remains identical to the source raster. When UserUnit is not recorded, the printed size will depends on the DPI value. If this parameter is not set, but DPI is specified, then it will default to NO (so that the printed size depends on the DPI value). If this parameter is not set and DPI is not specified, then UserUnit will be recorded (so that the raster size in pixels recognized by GDAL on reading remain identical to the source raster).
- **PREDICTOR=[1/2]:** Defaults to `1`. Only for DEFLATE compression. Might be set to 2 to use horizontal predictor that can make files smaller (but not always!).
- **JPEG_QUALITY=1-100:** Defaults to `75`. Set the JPEG quality when using JPEG compression. A value of 100 is best quality (least compression), and 1 is worst quality (best compression).
- **JPEG2000_DRIVER=[JP2KAK/JP2ECW/JP2OpenJPEG/JPEG2000]:** Set the JPEG2000 driver to use. If not specified, it will be searched in the previous list.
- **TILED=[YES/NO]:** Defaults to `NO`. By default monoblock files are created. This option can be used to force creation of tiled PDF files.
- **BLOCKXSIZE=value:** Defaults to `256`. Sets tile width.
- **BLOCKYSIZE=value:** Defaults to `256`. Set tile height.
- **CLIPPING_EXTENT=xmin,ymin,xmax,ymax:** Set the clipping extent for the main source dataset and for the optional extra rasters. The coordinates are expressed in the units of the SRS of the dataset. If not specified, the clipping extent is set to the extent of the main source dataset.
- **LAYER_NAME=value:** Name for layer where the raster is placed. If specified, the raster will be be placed into a layer that can be toggled/un-toggled in the "Layer tree" of the PDF reader.
- **EXTRA_RASTERS=<dataset_ids>:** Comma separated list of georeferenced rasters to insert into the page. Those rasters are displayed on top of the main source raster. They must be georeferenced in the same projection,

and they will be clipped to CLIPPING_EXTENT if it is specified (otherwise to the extent of the main source raster).

- **EXTRA_RASTERS_LAYER_NAME=<dataset_names>:** Comma separated list of name for each raster specified in EXTRA_RASTERS. If specified, each extra raster will be be placed into a layer, named with the specified value, that can be toggled/un-toggled in the "Layer tree" of the PDF reader. If not specified, all the extra rasters will be placed in the default layer.
- **EXTRA_STREAM=<content>:** A PDF content stream to draw after the imagery, typically to add some text. It may refer to any of the 14 standard PDF Type 1 fonts (omitting hyphens), as /FTimesRoman, /FTimesBold, /FHelvetica, /FCourierOblique, ... , in which case the required resource dictionary will be inserted.
- **EXTRA_IMAGES=<image_file_name,x,y,scale[,link=some_url]> (possibly repeated):** A list of (ungeoreferenced) images to insert into the page as extra content. This is useful to insert logos, legends, etc... x and y are in user units from the lower left corner of the page, and the anchor point is the lower left pixel of the image. scale is a magnifying ratio (use 1 if unsure). If link=some_url is specified, the image will be selectable and its selection will cause a web browser to be opened on the specified URL.
- **EXTRA_LAYER_NAME=value:** Name for layer where the extra content specified with EXTRA_STREAM or EXTRA_IMAGES is placed. If specified, the extra content will be be placed into a layer that can be toggled/un-toggled in the "Layer tree" of the PDF reader.
- **MARGIN=value:** Margin around image in user units.
- **LEFT_MARGIN=value:** Margin left of image in user units.
- **RIGHT_MARGIN=value:** Margin right of image in user units.
- **TOP_MARGIN=value:** Margin above image in user units.
- **BOTTOM_MARGIN=value:** Margin below image in user units.
- **GEO_ENCODING=[NONE/ISO32000]:** Defaults to `ISO32000`. Set the Geo encoding method to use. Support for deprecated OGC_BP and BOTH options has been removed in GDAL 3.11
- **NEATLINE=<polygon_definition_in_wkt>:** Set the NEATLINE to use.
- **XMP=[NONE/<xml_xmp_content>]:** By default, if the source dataset has data in the 'xml:XMP' metadata domain, this data will be copied to the output PDF, unless this option is set to NONE. The XMP xml string can also be directly set to this option.
- **WRITE_INFO=[YES/NO]:** Defaults to `YES`. By default, the AUTHOR, CREATOR, CREATION_DATE, KEYWORDS, PRODUCER, SUBJECT and TITLE information will be written into the PDF Info block from the corresponding metadata item from the source dataset, or if not set, from the corresponding creation option. If this option is set to NO, no information will be written.
- **AUTHOR=value:** Author metadata to write into the PDF Info block.
- **CREATOR=value:** Creator metadata to write into the PDF Info block.
- **CREATION_DATE=value:** Create date metadata to write into the PDF Info block. The format of the value must be D:YYYYMMDDHHmmSSOHH'mm' (e.g. D:20121122132447+02'00' for 22 nov 2012 13:24:47 GMT+02) (see PDF Reference, version 1.7, page 160)
- **KEYWORDS=value:** Keywords metadata to write into the PDF Info block.
- **PRODUCER=value:** Producer metadata to write into the PDF Info block.
- **SUBJECT=value:** Subject metadata to write into the PDF Info block.
- **TITLE=value:** Title metadata to write into the PDF Info block.
- **OGR_DATASOURCE=value:** Name of the OGR datasource to display on top of the raster layer.

- **OGR_DISPLAY_FIELD=value:** Name of the field (matching the name of a field from the OGR layer definition) to use to build the label of features that appear in the "Model Tree" UI component of a well-known PDF viewer. For example, if the OGR layer has a field called "ID", this can be used as the value for that option : features in the "Model Tree" will be labelled from their value for the "ID" field. If not specified, sequential generic labels will be used ("feature1", "feature2", etc...).
- **OGR_DISPLAY_LAYER_NAMES=value:** Comma separated list of names to display for the OGR layers in the "Model Tree". This option is useful to provide custom names, instead of OGR layer name that are used when this option is not specified. When specified, the number of names should be the same as the number of OGR layers in the datasource (and in the order they appear when listed by ogrinfo for example).
- **OGR_WRITE_ATTRIBUTES=[YES/NO]:** Defaults to `YES`. Whether to write attributes of OGR features.
- **OGR_LINK_FIELD=value:** Name of the field (matching the name of a field from the OGR layer definition) to use to cause clicks on OGR features to open a web browser on the URL specified by the field value.
- **OFF_LAYERS=value:** Comma separated list of layer names that should be initially hidden. By default, all layers are visible. The layer names can come from LAYER_NAME (main raster layer name), EXTRA_RASTERS_LAYER_NAME, EXTRA_LAYER_NAME and OGR_DISPLAY_LAYER_NAMES.
- **EXCLUSIVE_LAYERS=value:** Comma separated list of layer names, such that only one of those layers can be visible at a time. This is the behavior of radio-buttons in a graphical user interface. The layer names can come from LAYER_NAME (main raster layer name), EXTRA_RASTERS_LAYER_NAME, EXTRA_LAYER_NAME and OGR_DISPLAY_LAYER_NAMES.
- **JAVASCRIPT=<script>:** Javascript content to run at document opening. See Acrobat(R) JavaScript Scripting Reference.
- **JAVASCRIPT_FILE=<filename>:** Name of Javascript file to embed and run at document opening. See Acrobat(R) JavaScript Scripting Reference.
- **COMPOSITION_FILE=<xml_filename>:** (GDAL >= 3.0) See below paragraph "Creation of PDF file from a XML composition file"

6.116.8 Update of existing files

Existing PDF files (created or not with GDAL) can be opened in update mode in order to set or update the following elements :

- Geotransform and associated projection (with SetGeoTransform() and SetProjection())
- GCPs (with SetGCPs())
- Neatline (with SetMetadataItem("NEATLINE", polygon_definition_in_wkt))
- Content of Info object (with SetMetadataItem(key, value) where key is one of AUTHOR, CREATOR, CREATION_DATE, KEYWORDS, PRODUCER, SUBJECT and TITLE)
- xml:XMP metadata (with SetMetadata(md, "xml:XMP"))

For geotransform or GCPs, the Geo encoding method used by default is ISO32000.

Updated elements are written at the end of the file, following the incremental update method described in the PDF specification.

6.116.9 Creation of PDF file from a XML composition file (GDAL >= 3.0)

A PDF file can be generate from a XML file that describes the composition of the PDF:

- number of pages
- layer tree, with visibility state, exclusion groups

- definition or 0, 1 or several georeferenced areas per page
- page content made of rasters, vectors or labels

The GDALCreate() API must be used with width = height = bands = 0 and datatype = GDT_Unknown and *COMPOSITION_FILE* must be the single creation option.

The XML schema against which the composition file must validate is pdfcomposition.xsd

Example on how to use the API:

```
char** papszOptions = CSLSetNameValue(nullptr, "COMPOSITION_FILE", "the.xml");
GDALDataset* ds = GDALCreate("the.pdf", 0, 0, 0, GDT_Unknown, papszOptions);
// return a non-null (fake) dataset in case of success, nullptr otherwise.
GDALClose(ds);
CSLDestroy(papszOptions);
```

A sample Python script gdal_create_pdf.py is also available. Starting with GDAL 3.2, the *gdal_create* utility can also be used.

Example of a composition XML file:

```
<PDFComposition>
    <Metadata>
        <Author>Even</Author>
    </Metadata>

    <LayerTree displayOnlyOnVisiblePages="true">
        <Layer id="l1" name="Satellite imagery"/>
        <Layer id="l2" name="OSM data">
            <Layer id="l2.1" name="Roads" initiallyVisible="false"/>
            <Layer id="l2.2" name="Buildings" mutuallyExclusiveGroupId="group1">
                <Layer id="l2.2.text" name="Buildings name"/>
            </Layer>
            <Layer id="l2.3" name="Cadastral parcels" mutuallyExclusiveGroupId="group1"/>
        </Layer>
    </LayerTree>

    <Page id="page_1">
        <DPI>72</DPI>
        <Width>10</Width>
        <Height>15</Height>
        <Georeferencing id="georeferenced">
            <SRS dataAxisToSRSAxisMapping="2,1">EPSG:4326</SRS>
            <BoundingBox x1="1" y1="1" x2="9" y2="14"/>
            <BoundingPolygon>POLYGON((1 1,9 1,9 14,1 14,1 1))</BoundingPolygon>
            <ControlPoint x="1"  y="1"  GeoY="48"  GeoX="2"/>
            <ControlPoint x="1"  y="14" GeoY="49"  GeoX="2"/>
            <ControlPoint x="9"  y="1"  GeoY="49"  GeoX="3"/>
            <ControlPoint x="9"  y="14" GeoY="48"  GeoX="3"/>
        </Georeferencing>

        <Content>
            <IfLayerOn layerId="l1">
                <!-- image drawn, and stretched to (x1,y1)->(x2,y2), without reading its␣
↪georeferencing -->
```

(continues on next page)

(continued from previous page)

```
                <Raster dataset="satellite.png" x1="1" y1="1" x2="9" y2="14"/>
            </IfLayerOn>
            <IfLayerOn layerId="l2">
                <IfLayerOn layerId="l2.1">
                    <Raster dataset="roads.jpg" x1="1" y1="1" x2="9" y2="14"/>
                    <!-- vector drawn with coordinates in PDF coordinate space -->
                    <Vector dataset="roads_pdf_units.shp" layer="roads_pdf_units"␣
↪visible="false">
                        <LogicalStructure displayLayerName="Roads" fieldToDisplay="road_
↪name"/>>
                    </Vector>
                </IfLayerOn>
                <IfLayerOn layerId="l2.2">
                    <!-- image drawn by taking into account its georeferencing -->
                    <Raster dataset="buildings.tif" georeferencingId="georeferenced"/>
                    <IfLayerOn layerId="l2.2.text">
                        <!-- vector drawn by taking into account its georeferenced␣
↪coordinates -->
                        <VectorLabel dataset="labels.shp" layer="labels"␣
↪georeferencingId="georeferenced">
                        </VectorLabel>
                    </IfLayerOn>
                </IfLayerOn>
                <IfLayerOn layerId="l2.3">
                    <PDF dataset="parcels.pdf">
                        <Blending function="Normal" opacity="0.7"/>
                    </PDF>
                </IfLayerOn>
            </IfLayerOn>
        </Content>
    </Page>

    <Page id="page_2">
        <DPI>72</DPI>
        <Width>10</Width>
        <Height>15</Height>
        <Content>
        </Content>
    </Page>

    <Outline>
        <OutlineItem name="turn only layer 'Satellite imagery' on, and switch to␣
↪fullscreen" italic="true" bold="true">
            <Actions>
                <SetAllLayersStateAction visible="false"/>
                <SetLayerStateAction visible="true" layerId="l1"/>
                <JavascriptAction>app.fs.isFullScreen = true;</JavascriptAction>
            </Actions>
        </OutlineItem>
        <OutlineItem name="Page 1" pageId="page_1">
            <OutlineItem name="Important feature !">
                <Actions>
```

(continues on next page)

(continued from previous page)

```
                    <GotoPageAction pageId="page_1" x1="1" y1="2" x2="3" y2="4"/>
                </Actions>
            </OutlineItem>
        </OutlineItem>
        <OutlineItem name="Page 2" pageId="page_2"/>
    </Outline>

</PDFComposition>
```

6.116.10 Build dependencies

For read support, GDAL must be built against one of the following libraries :

- Poppler (GPL-licensed)
- PoDoFo (LGPL-licensed)
- PDFium (New BSD-licensed)

Note: it is also possible to build against a combination of several of the above libraries. PDFium will be used in priority over Poppler, itself used in priority over PoDoFo.

6.116.10.1 Poppler

libpoppler itself must have been configured with -DENABLE_UNSTABLE_API_ABI_HEADERS=ON so that the xpdf C++ headers are available. Note: the poppler C++ API isn't stable, so the driver compilation may fail with too old or too recent poppler versions.

6.116.10.2 PoDoFo

As a partial alternative, the PDF driver can be compiled against libpodofo to avoid the libpoppler dependency. This is sufficient to get the georeferencing and vector information. However, for getting the imagery, the pdftoppm utility that comes with the poppler distribution must be available in the system PATH. A temporary file will be generated in a directory determined by the following configuration options : *CPL_TMPDIR*, `TMPDIR` or `TEMP` (in that order). If none are defined, the current directory will be used. Successfully tested versions are libpodofo 0.8.4, 0.9.1 and 0.9.3. Important note: using PoDoFo 0.9.0 is strongly discouraged, as it could cause crashes in GDAL due to a bug in PoDoFo.

6.116.10.3 PDFium

Using PDFium as a backend allows access to raster, vector, georeferencing and other metadata. The PDFium backend has also support for arbitrary overviews, for fast zoom-out.

Only GDAL builds against static builds of PDFium have been tested. Building PDFium can be challenging, and particular builds must be used to work properly with GDAL.

With GDAL >= 3.13

The scripts in the https://github.com/rouault/pdfium_build_gdal_3_13 repository must be used to build a patched version of PDFium.

Note that it requires a C++20 compiler (supporting C++20 "concepts")

With GDAL = 3.12

The scripts in the https://github.com/rouault/pdfium_build_gdal_3_12 repository must be used to build a patched version of PDFium.

Note that it requires a C++20 compiler (supporting C++20 "concepts")

With GDAL = 3.11

The scripts in the https://github.com/rouault/pdfium_build_gdal_3_11 repository must be used to build a patched version of PDFium.

With GDAL = 3.10

The scripts in the https://github.com/rouault/pdfium_build_gdal_3_10 repository must be used to build a patched version of PDFium.

With GDAL = 3.9

The scripts in the https://github.com/rouault/pdfium_build_gdal_3_9 repository must be used to build a patched version of PDFium.

With GDAL = 3.8

The scripts in the https://github.com/rouault/pdfium_build_gdal_3_8 repository must be used to build a patched version of PDFium.

With GDAL 3.7

The scripts in the https://github.com/rouault/pdfium_build_gdal_3_7 repository must be used to build a patched version of PDFium.

With GDAL 3.5 and 3.6

The scripts in the https://github.com/rouault/pdfium_build_gdal_3_5 repository must be used to build a patched version of PDFium.

With GDAL 3.4

The scripts in the https://github.com/rouault/pdfium_build_gdal_3_4 repository must be used to build a patched version of PDFium.

With GDAL 3.2 and 3.3

The scripts in the https://github.com/rouault/pdfium_build_gdal_3_2 repository must be used to build a patched version of PDFium.

With GDAL 3.1.x

The scripts in the https://github.com/rouault/pdfium_build_gdal_3_1 repository must be used to build a patched version of PDFium.

6.116.11 Examples

- Create a PDF from 2 rasters (main_raster and another_raster), such that main_raster is initially displayed, and they are exclusively displayed :

```
gdal_translate -of PDF main_raster.tif my.pdf -co LAYER_NAME=main_raster
               -co EXTRA_RASTERS=another_raster.tif -co EXTRA_RASTERS_LAYER_
↪NAME=another_raster
               -co OFF_LAYERS=another_raster -co EXCLUSIVE_LAYERS=main_raster,
↪another_raster
```

- Create of PDF with some JavaScript :

```
gdal_translate -of PDF my.tif my.pdf -co JAVASCRIPT_FILE=script.js
```

where script.js is :

```
button = app.alert({cMsg: 'This file was generated by GDAL. Do you want to visit␣
↪its website ?', cTitle: 'Question', nIcon:2, nType:2});
if (button == 4) app.launchURL('http://gdal.org/');
```

6.116.12 See also

PDF vector documentation page

Specifications :

- Adobe Supplement to ISO 32000
- PDF Reference, version 1.7
- Acrobat(R) JavaScript Scripting Reference
- OGC PDF Georegistration Encoding Best Practice Version 2.2 (08-139r3) (not recommended)

Libraries :

- Poppler homepage
- PoDoFo homepage
- PDFium homepage
- PDFium forked version for simpler builds

Samples :

- A few Geospatial PDF samples
- Tutorial to generate Geospatial PDF maps from OSM data

6.117 PDS -- Planetary Data System v3

Driver short name

PDS

Driver built-in by default

This driver is built-in by default

PDS is a format used primarily by NASA to store and distribute solar, lunar and planetary imagery data. GDAL provides read-only access to PDS formatted imagery data.

PDS files often have the extension .img, sometimes with an associated .lbl (label) file. When a .lbl file exists it should be used as the dataset name rather than the .img file.

In addition to support for most PDS imagery configurations, this driver also reads georeferencing and coordinate system information as well as selected other header metadata.

Implementation of this driver was supported by the United States Geological Survey.

Note

PDS3 datasets can incorporate a VICAR header. By default, GDAL will use the PDS driver in that situation. Starting with GDAL 3.1, if the *GDAL_TRY_PDS3_WITH_VICAR* configuration option is set to YES, the dataset will be opened by the *VICAR* driver.

6.117.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.117.2 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). The following configuration option is available:

- **GDAL_TRY_PDS3_WITH_VICAR=[YES/NO]:** (GDAL >= 3.1) Defaults to `NO`. If `YES`, open the dataset with the *VICAR* driver instead.

6.117.3 Georeferencing

Due to ambiguities in the PDS specification, the georeferencing of some products is subtly or grossly incorrect. There are configuration variables which may be set for these products to correct the interpretation of the georeferencing. Some details are available in ticket #5941 and ticket #3940.

As a test, download both the label and image for the lunar LOLA DEM (digital elevation file) LOLA PDS label and LOLA PDS v3 image. Using gdalinfo, the reported centered should be perfectly at 0.0, 0.0 meters in Cartesian space without any configuration options.

$ gdalinfo ldem_4.lbl

Example conversion to GeoTiff:

$ gdal_translate ldem_4.lbl out_LOLA.tif

Example conversion and applying offset and multiplier values as defined in some PDS labels:

$ gdal_translate -ot Float32 -unscale ldem_4.lbl out_LOLA_32bit.tif

---

To show an example to correct an offset issue we can use the MOLA DEM from the PDS. Download both the MOLA PDS label and MOLA PDS v3 image. The MOLA labels currently contain a one pixel offset. To read this file correctly, use GDAL with these options.

$ gdalinfo --config PDS_SampleProjOffset_Shift -0.5 --config PDS_LineProjOffset_Shift -0.5 megt90n000cb.lbl

Again with these optional parameters, the center should be perfectly 0.0, 0.0 meters in Cartesian space.

Example conversion for MOLA:

$ gdal_translate --config PDS_SampleProjOffset_Shift -0.5 --config PDS_LineProjOffset_Shift -0.5 megt90n000cb.lbl out_MOLA_4ppd.tif

Example conversion and applying offset and multiplier values as defined in some PDS labels:

$ gdal_translate -ot Float32 -unscale --config PDS_SampleProjOffset_Shift -0.5 --config PDS_LineProjOffset_Shift -0.5 megt90n000cb.lbl out_MOLA_4ppd_32bit.tif

---

PDS is part of a family of related formats including ISIS2 and ISIS3.

6.117.4 See Also

- Implemented as pdsdataset.cpp.
- NASA Planetary Data System
- *ISIS2 -- USGS Astrogeology ISIS Cube (Version 2)* driver.
- *ISIS3 -- USGS Astrogeology ISIS Cube (Version 3)* driver.

6.118 PDS4 -- NASA Planetary Data System (Version 4)

Driver short name

PDS4

Driver built-in by default

This driver is built-in by default

PDS4 is a format used primarily by NASA to store and distribute solar, lunar and planetary imagery data. GDAL provides read-write access to PDS4 formatted imagery data.

PDS4 files are compose of a .xml (label) file which references a raw imagery file. The driver also supports imagery stored in a separate uncompressed GeoTIFF file with a strip organization compatible with a raw imagery file.

The driver also reads and write georeferencing and coordinate system information as well as selected other header metadata.

A mask band is attached to each source band. The value of this mask band is 0 when the pixel value is one of the missing constants.

Implementation of this driver was supported by the United States Geological Survey.

PDS4 is part of a family of related formats including PDS and ISIS3.

The PDS4 driver supports reading and writing ASCII fixed-with, binary fixed-with and delimited(CSV) tables as OGR vector layers.

6.118.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.118.2 Metadata

The PDS4 label can be retrieved as XML-serialized content in the xml:PDS4 metadata domain.

On creation, a source template label can be passed to the SetMetadata() interface in the "xml:PDS4" metadata domain.

6.118.3 Open options (vector only)

Added in version 3.0.

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). When opening a PDS4 vector dataset, the following open options are available:

- **LAT=value:** Defaults to `Latitude`. Name of a field containing a Latitude value.
- **LONG=value:** Defaults to `Longitude`. Name of a field containing a Longitude value.
- **ALT=value:** Name of a field containing a Altitude value. Defaults to Altitude.
- **WKT=value:** Name of a field containing a WKT value.
- **KEEP_GEOM_COLUMNS=[YES/NO]:** Defaults to `NO`. Whether to expose original x/y/geometry columns as regular fields.

6.118.4 Creation support

The PDS4 driver supports updating imagery of existing datasets, creating new datasets through the CreateCopy() and Create() interfaces.

When using CreateCopy(), gdal_translate or gdalwarp, an effort is made to preserve as much as possible of the original label when doing PDS4 to PDS4 conversions. This can be disabled with the *USE_SRC_LABEL=NO* creation option.

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following dataset creation options are available:

- Raster only:
  - **APPEND_SUBDATASET=value:** See *Subdataset / multiple image support*.
  - **IMAGE_FILENAME=<filename>:** Override default external image filename.
  - **IMAGE_EXTENSION=value:** Override default extension of the external image filename. The default is 'img' for *IMAGE_FORMAT=RAW* or 'tif' for *IMAGE_FORMAT=GEOTIFF*.
  - **IMAGE_FORMAT=[RAW/GEOTIFF]:** Format of the image file. If using RAW, the imagery is put in a raw file whose filename is the main filename with a .img extension. If using GEOTIFF, the imagery is put in a separate GeoTIFF file, whose filename is the main filename with a .tif extension. Defaults to RAW
  - **INTERLEAVE=[BSQ/BIP/BIL]:** Defaults to `BSQ`. Pixel organization in the image file. BSQ is Band SeQuential, BIP is Band Interleaved per Pixel and BIL is Band Interleave Per Line. BIL is not valid for *IMAGE_FORMAT=GEOTIFF*. Starting with GDAL 3.5, when copying from a source dataset with multiple bands which advertises a INTERLEAVE metadata item, if the INTERLEAVE creation option is not specified, the source dataset INTERLEAVE will be automatically taken into account. See *Multiband pixel organization (INTERLEAVE metadata item)* for more details.
  - **USE_SRC_LABEL=[YES/NO]:** Defaults to `YES`. Whether to use the source label in PDS4 to PDS4 conversions.
  - **ARRAY_TYPE=[Array/Array_2D/Array_2D_Image/Array_2D_Map/Array_2D_Spectrum/Array_3D/Array_3D_Image/Array_3D_Movie/Array_3D_Spectrum]:** Defaults to `Array_3D_Image`. To set the XML element that defines the type of array. Using a Array_2D* for a multiband image is not supported. When using a Array_2D* value, INTERLEAVE will be ignored.
  - **ARRAY_IDENTIFIER=<string>:** (GDAL >= 3.0) Identifier to put in the Array element.
  - **UNIT=<string>:** (GDAL >= 3.0) Content of the Element_Array.unit. If not provided, the unit of the source band in case of copying from another raster will be used (if present on the source band).
  - **CREATE_LABEL_ONLY=[YES/NO]:** (GDAL >= 3.1) If set to YES, and used in a gdal_translate / CreateCopy() context where the source dataset is a ENVI, GeoTIFF, ISIS3, VICAR, FITS or PDS3 dataset, whose layout is compatible with a raw binary format, as supported by PDS4, then only the label XML file will be generated, and it will reference the raw binary file of the source dataset. The IMAGE_FILENAME, IMAGE_FORMAT and INTERLEAVE creation options are ignored in that situation.
- Raster and vector:
  - **VAR_***=string. If options like VAR_XXXX=yyyy are specified, any {XXXX} string in the template label will be replaced by the yyyy value.
  - **TEMPLATE=<filename>:** Template label to use. If not specified and not creating from an existing PDS4 file, the data/pds4_template.xml file will be used. For GDAL utilities to find this default PDS4 template, GDAL's data directory should be defined in your environment (typically on Windows builds).
  - **LATITUDE_TYPE=[Planetocentric/Planetographic]:** Defaults to `Planetocentric`. Value of latitude_type.

- **LONGITUDE_DIRECTION=[Positive East/Positive West.]:** Defaults to `Positive` East. Value of longitude_direction.
- **RADII=semi_major_radius,semi_minor_radius>:** To override the ones of the SRS. Note that the first value (semi_major_radius) will be used to set the <pds:semi_major_radius> and <pds:semi_minor_radius> XML elements, and that second value (semi_minor_radius) will be used to set the <pds:polar_radius> XML element.
- **BOUNDING_DEGREES=west_lon,south_lat,east_lon,north_lat:** Manually set bounding box
- **PROPAGATE_SRC_METADATA=[YES/NO]:** (GDAL >= 3.12) Defaults to `YES`. Whether to propagate particular metadata domains, such as json:ISIS3. When YES (the default), if IMAGE_FORMAT=GEOTIFF, that metadata is also written into the GeoTIFF file.

6.118.5 Layer creation options (vector/table datasets)

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALDatasetCreateLayer()`* (C) or *`Dataset.CreateLayer`* (Python). (Starting with GDAL 3.0) When creating a PDS4 vector dataset, or appending a new table to an existing table, the following layer creation options are available:

- **TABLE_TYPE=[DELIMITED/CHARACTER/BINARY.]:** Determines the type of the PDS4 table to create. DELIMITED is the default and corresponds to a CSV table file (with comma field separator). CHARACTER corresponds to a fixed-width ASCII table. BINARY corresponds to a fixed-width table. For fixed-width table, for String fields, an arbitrary width of 64 bytes is used if there is no explicit field set in the OGR field definition. Only DELIMITED supports arbitrary encoding of geometry as a WKT string. The two other table types only support points for geographic coordinates (LAT, LONG).
- **LINE_ENDING=[CRLF/LF]:** (GDAL >= 3.4) Defaults to `CRLF`. Determines the line-ending character sequence. Only applies if *`TABLE_TYPE`* is DELIMITED or CHARACTER.
- **GEOM_COLUMNS=[AUTO/WKT/LONG_LAT]:** Specify how the geometry is encoded. In AUTO mode, for DELIMITED tables, if the input geometry is Point with a geographic CRS attached to the laye, then a LONG and LAT columns will be created to store the point coordinates. For other geometry types, a WKT column is used. The WKT value of this option can also be used to force a WKT column to be created when a LONG and LAT columns would have been possible. For fixed-width table types, only AUTO and LONG_LAT are possible.
- **CREATE_VRT=[YES/NO]:** Defaults to YES for a DELIMITED table. In that case, a OGR VRT (XML file) will be created along-side the .csv file.
- **LAT=value:** Defaults to `Latitude`. Name of a field containing a Latitude value. Only used when the geometry comes from a Point layer with geographic CRS
- **LONG=value:** Defaults to `Longitude`. Name of a field containing a Longitude value. Only used when the geometry comes from a Point layer with geographic CRS
- **ALT=value:** Defaults to `Altitude`. Name of a field containing a Altitude value. Only used when the geometry comes from a Point layer with geographic CRS
- **WKT=value:** Name of a field containing a WKT value.

Table files are created in the same directory as the XML file, using its name as a prefix. For example if creating a "foo.xml" PDS4 dataset and a "bar" layer, a "foo_bar.dat" file will be created.

6.118.6 Subdataset / multiple image support

If several Array objects are present in the label, they will be reported as separate subdatasets (typically the main subdataset is an Array3D, and backplanes are represented as Array2D).

Since GDAL 3.0, creation of new datasets with subdatasets is supported (through the *`APPEND_SUBDATASET=YES`* creation option). One important restriction is that, given that the georeferencing information in the PDS4 XML label is

global for the whole dataset, all subdatasets must share the same georeferencing information: coordinate reference system, georegistration and resolution. Appending to both RAW and GEOTIFF raster is supported. In append mode, most creation options are ignored, except *INTERLEAVE* (if GeoTIFF output image), *ARRAY_TYPE* and *ARRAY_IDENTIFIER*.

6.118.7 PDS4 raster examples

Listing bands and subdatasets:

```
$ gdalinfo b0011_p237201_01_01v02.xml

Driver: PDS4/NASA Planetary Data System 4
Files: b0011_p237201_01_01v02.xml
       b0011_p237201_01_01v02.qub
Size is 512, 512
Coordinate System is `'
Image Structure Metadata:
  INTERLEAVE=BAND
Subdatasets:
  SUBDATASET_1_NAME=PDS4:b0011_p237201_01_01v02.xml:1:1
  SUBDATASET_1_DESC=Image file b0011_p237201_01_01v02.qub, array Spectral_Qube_Object
  SUBDATASET_2_NAME=PDS4:b0011_p237201_01_01v02.xml:1:2
  SUBDATASET_2_DESC=Image file b0011_p237201_01_01v02.qub, array iof_r2
  SUBDATASET_3_NAME=PDS4:b0011_p237201_01_01v02.xml:1:3
  SUBDATASET_3_DESC=Image file b0011_p237201_01_01v02.qub, array iof_r7
  SUBDATASET_4_NAME=PDS4:b0011_p237201_01_01v02.xml:1:4
[...]
  SUBDATASET_16_DESC=Image file b0011_p237201_01_01v02.qub, array emission_angle
  SUBDATASET_17_NAME=PDS4:b0011_p237201_01_01v02.xml:1:17
  SUBDATASET_17_DESC=Image file b0011_p237201_01_01v02.qub, array phase_angle
  SUBDATASET_18_NAME=PDS4:b0011_p237201_01_01v02.xml:1:18
  SUBDATASET_18_DESC=Image file b0011_p237201_01_01v02.qub, array approx_incidence_angle
  SUBDATASET_19_NAME=PDS4:b0011_p237201_01_01v02.xml:1:19
  SUBDATASET_19_DESC=Image file b0011_p237201_01_01v02.qub, array approx_emission_angle
  SUBDATASET_20_NAME=PDS4:b0011_p237201_01_01v02.xml:1:20
  SUBDATASET_20_DESC=Image file b0011_p237201_01_01v02.qub, array approx_phase_angle
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0,  512.0)
Upper Right (  512.0,    0.0)
Lower Right (  512.0,  512.0)
Center      (  256.0,  256.0)
Band 1 Block=512x1 Type=Int16, ColorInterp=Undefined
  Offset: 0.146998785514825,   Scale:4.48823844390647e-06
Band 2 Block=512x1 Type=Int16, ColorInterp=Undefined
  Offset: 0.146998785514825,   Scale:4.48823844390647e-06
Band 3 Block=512x1 Type=Int16, ColorInterp=Undefined
  Offset: 0.146998785514825,   Scale:4.48823844390647e-06
Band 4 Block=512x1 Type=Int16, ColorInterp=Undefined
  Offset: 0.146998785514825,   Scale:4.48823844390647e-06
Band 5 Block=512x1 Type=Int16, ColorInterp=Undefined
  Offset: 0.146998785514825,   Scale:4.48823844390647e-06
```

The information displayed by default is the one of the first subdataset (SUBDATASET_1_NAME)

Getting information on a subdataset:

```
$ gdalinfo PDS4:b0011_p237201_01_01v02.xml:1:2

Driver: PDS4/NASA Planetary Data System 4
Files: b0011_p237201_01_01v02.xml
       b0011_p237201_01_01v02.qub
Size is 512, 512
Coordinate System is `'
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0,  512.0)
Upper Right (  512.0,    0.0)
Lower Right (  512.0,  512.0)
Center      (  256.0,  256.0)
Band 1 Block=512x1 Type=Int16, ColorInterp=Undefined
  Offset: 0.04984971,   Scale:7.454028e-06
```

Conversion to GeoTIFF of a given subdatasets:

```
$ gdal_translate PDS4:b0011_p237201_01_01v02.xml:1:2 iof_r2.tif
```

Conversion to GeoTIFF of a all subdatasets:

```
$ gdal_translate -sds b0011_p237201_01_01v02.xml b0011_p237201_01_01v02.tif
```

This will create b0011_p237201_01_01v02_X.tif files where X=1,....,N

Creation of a new PDS4 dataset, using the default template and setting its parameterized variables:

```
$ gdal_translate input.tif output.xml -of PDS4 \
            -co VAR_TARGET_TYPE=Satellite \
            -co VAR_TARGET=Moon \
            -co VAR_OBSERVING_SYSTEM_NAME=LOLA \
            -co VAR_LOGICAL_IDENTIFIER=Lunar_LRO_LOLA_DEM_Global_64ppd.tif \
            -co VAR_TITLE="LRO LOLA Digital Elevation Model (DEM) 64ppd" \
            -co VAR_INVESTIGATION_AREA_NAME="Lunar Reconnaissance Orbiter" \
            -co VAR_INVESTIGATION_AREA_LID_REFERENCE="urn:nasa:pds:context:instrument_
↪host:spacecraft.lro"
```

Creation of the same PDS4 dataset as above, using the default template but setting its parameterized variables from a text file. Helps with long command lines:

Create a text file "myOptions.txt" with the below content

```
#This is a comment
#Conversion parameters for the LRO LOLA dataset
-co VAR_TARGET_TYPE=Satellite
-co VAR_TARGET=Moon
-co VAR_OBSERVING_SYSTEM_NAME=LOLA
-co VAR_LOGICAL_IDENTIFIER=Lunar_LRO_LOLA_DEM_Global_64ppd.tif
-co VAR_TITLE="LRO LOLA Digital Elevation Model (DEM) 64ppd"
-co VAR_INVESTIGATION_AREA_NAME="Lunar Reconnaissance Orbiter"
-co VAR_INVESTIGATION_AREA_LID_REFERENCE="urn:nasa:pds:context:instrument_
↪host:spacecraft.lro"
#end of file
```

```
gdal_translate input.tif output.xml -of PDS4 --optfile myOptions.txt
```

For more on --optfile, consult the general documentation on GDAL utilities.

Creation of a PDS4 dataset, using a non default template (here on a HTTP server, but local filename also possible):

```
$ gdal_translate input.tif output.xml -of PDS4 \
            -co TEMPLATE=http://example.com/mytemplate.xml
```

Creation of a PDS4 dataset from a source PDS4 dataset (using the XML file of this source PDS4 dataset as an implicit template), with subsetting:

```
$ gdal_translate input.xml output.xml -of PDS4 -projwin ullx ully lrx lry
```

In Python, creation of a PDS4 dataset from a GeoTIFF, using a base template into which one substitute one element with a new value:

```
from osgeo import gdal
from lxml import etree

# Customization of template
template = open('template.xml','rb').read()
root = etree.XML(template)
ns = '{http://pds.nasa.gov/pds4/pds/v1}'
identifier = root.find(".//{ns}Identification_Area/{ns}logical_identifier".format(ns =␣
↪ns))
identifier.text = 'new_identifier'

# Serialize the modified template in a in-memory file
in_memory_template = '/vsimem/template.xml'
gdal.FileFromMemBuffer(in_memory_template, etree.tostring(root))

# Create the output dataset
gdal.Translate('out.xml', 'in.tif', format = 'PDS4',
               creationOptions = ['TEMPLATE='+in_memory_template])

# Cleanup
gdal.Unlink(in_memory_template)
```

Appending a new image (subdataset) to an existing PDS4 dataset.

```
$ gdal_translate new_image.tif existing_output.xml -of PDS4 \
                      -co APPEND_SUBDATASET=YES \
                      -co ARRAY_IDENTIFIER=my_new_image
```

Adding a PDS4 label to an existing ISIS3 dataset. (GDAL >= 3.1)

```
$ gdal_translate dataset.cub dataset.xml -of PDS4 -co CREATE_LABEL_ONLY=YES
```

6.118.8 PDS4 vector examples

Displaying the content of a PDS4 dataset with a table:

```
$ ogrinfo -al my_pds4.xml
```

Converting a PDS4 dataset with a table to shapefile, by specifying columns that contain longitude and latitude:

```
$ ogr2ogr out.shp my_pds4.xml -oo LAT=my_lat_column -oo LONG=my_long_column
```

Converting a shapefile to a PDS4 dataset with a CSV-delimited table (with an implicit WKT column to store the geometry):

```
$ ogr2ogr my_out_pds4.xml in.shp
```

6.118.9 See Also:

- Implemented as pds4dataset.cpp.
- Official documentation
- Schemas, including the cartography extension
- *PDS -- Planetary Data System v3* driver.
- *ISIS3 -- USGS Astrogeology ISIS Cube (Version 3)* driver.

6.119 PLMosaic (Planet Labs Mosaics API)

Driver short name

PLMosaic

Build dependencies

libcurl

This driver can connect to Planet Labs Mosaics API. GDAL/OGR must be built with Curl support in order for the PLMosaic driver to be compiled.

The driver supports listing mosaics and reading them. Mosaics are accessed at their highest resolution. Mosaics are typically composed of quads of 4096x4096 pixels.

For mosaics of type Byte, overviews are available by using the tile API. For other data types, there is no support for overviews, so requests that involve downsampling may take a long time to complete.

6.119.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

6.119.2 Dataset name syntax

The minimal syntax to open a datasource is :

```
PLMosaic:[options]
```

Additional optional parameters can be specified after the ':' sign. Currently the following one is supported :

- **api_key**=value: To specify the Planet API key. It is mandatory, unless it is supplied through the open option API_KEY, or the configuration option PL_API_KEY.
- **mosaic**=mosaic_name: To specify the mosaic name.
- **cache_path**=path: To specify the path to a directory where cached quads (and tiles) are stored. A plmosaic_cache/{mosaic_name} subdirectory will be created under that path. The empty string can be used to disable any disk caching.
- **trust_cache**=YES/NO: Whether already cached quads should be reused directly, without prior checking if the server has a more recent version. Note: this only applies to quads, and not tiles. Default is NO.
- **use_tiles**=YES/NO: Whether to use the tile API to access full resolution data, instead of downloading quads. Only apply for Byte mosaics. Default is NO.

If several parameters are specified, they must be separated by a comma.

If no mosaic parameter is supplied, the list of available mosaics will be returned as subdatasets. If only one mosaic is available, it will be directly opened.

6.119.3 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are available : API_KEY, MOSAIC, CACHE_PATH, TRUST_CACHE and USE_TILES. They have the same semantics as the above-described parameters of same name.

6.119.4 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The following configuration options are available :

PL_API_KEY=value: To specify the Planet API key.

6.119.5 Location information

The special *Pixel_{x}_{y}* metadata item of the *LocationInfo* metadata domain, where x is the column and y is the line in the mosaic, can be queried to get information about the scenes that compose the underneath quad. This is the syntax used by the gdallocationinfo utility (see rfc-32)

Below an example of the return :

```
<LocationInfo>
  <Scenes>
    <Scene>
      <link>https://api.planet.com/data/v1/item-types/PSScene3Band/items/20161025_000336_
↪0e19</link>
    </Scene>
    <Scene>
      <link>https://api.planet.com/data/v1/item-types/PSScene3Band/items/20161119_000453_
↪0e14</link>
    </Scene>
    <Scene>
      <link>https://api.planet.com/data/v1/item-types/PSScene3Band/items/20161010_000309_
↪0e26</link>
    </Scene>
    <Scene>
```

(continues on next page)

(continued from previous page)

```
      <link>https://api.planet.com/data/v1/item-types/PSScene3Band/items/20161119_000452_
↪0e14</link>
    </Scene>
  </Scenes>
</LocationInfo>
```

6.119.5.1 Examples

Listing all mosaics available (with the rights of the account) :

```
gdalinfo "PLMosaic:" -oo API_KEY=some_value
```

or

```
gdalinfo "PLMosaic:api_key=some_value"
```

or

```
gdalinfo "PLMosaic:" --config PL_API_KEY some_value
```

returns (in case of multiple mosaics):

```
Driver: PLMOSAIC/Planet Labs Mosaics API
Files: none associated
Size is 512, 512
Coordinate System is `'
Image Structure Metadata:
  INTERLEAVE=PIXEL
Subdatasets:
  SUBDATASET_1_NAME=PLMOSAIC:mosaic=global_quarterly_2017q1_mosaic
  SUBDATASET_1_DESC=Mosaic global_quarterly_2017q1_mosaic
  ...
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0,  512.0)
Upper Right (  512.0,    0.0)
Lower Right (  512.0,  512.0)
Center      (  256.0,  256.0)
```

Open a particular mosaic :

```
gdalinfo "PLMosaic:mosaic=global_quarterly_2017q1_mosaic" -oo API_KEY=some_value
```

returns:

```
Driver: PLMOSAIC/Planet Labs Mosaics API
Files: none associated
Size is 8388608, 4427776
Coordinate System is:
PROJCS["WGS 84 / Pseudo-Mercator",
    GEOGCS["WGS 84",
        DATUM["WGS_1984",
            SPHEROID["WGS 84",6378137,298.257223563,
```

(continues on next page)

(continued from previous page)

```
                AUTHORITY["EPSG","7030"]],
            AUTHORITY["EPSG","6326"]],
        PRIMEM["Greenwich",0,
            AUTHORITY["EPSG","8901"]],
        UNIT["degree",0.0174532925199433,
            AUTHORITY["EPSG","9122"]],
        AUTHORITY["EPSG","4326"]],
    PROJECTION["Mercator_1SP"],
    PARAMETER["central_meridian",0],
    PARAMETER["scale_factor",1],
    PARAMETER["false_easting",0],
    PARAMETER["false_northing",0],
    UNIT["metre",1,
        AUTHORITY["EPSG","9001"]],
    AXIS["X",EAST],
    AXIS["Y",NORTH],
    EXTENSION["PROJ4","+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0␣
↪+y_0=0 +k=1.0 +units=m +nadgrids=@null +wktext +no_defs"],
    AUTHORITY["EPSG","3857"]]
Origin = (-20037508.342789243906736,13384429.400847502052784)
Pixel Size = (4.777314267823516,-4.777314267823516)
Metadata:
  FIRST_ACQUIRED=2017-01-01T00:00:00.000Z
  LAST_ACQUIRED=2017-04-01T00:00:00.000Z
  NAME=global_quarterly_2017q1_mosaic
Image Structure Metadata:
  INTERLEAVE=PIXEL
Corner Coordinates:
Upper Left  (-20037508.343,13384429.401) (180d 0' 0.00"W, 76d 0'57.94"N)
Lower Left  (-20037508.343,-7768448.059) (180d 0' 0.00"W, 57d 2'26.63"S)
Upper Right (20037508.343,13384429.401) (180d 0' 0.00"E, 76d 0'57.94"N)
Lower Right (20037508.343,-7768448.059) (180d 0' 0.00"E, 57d 2'26.63"S)
Center      (       0.000, 2807990.671) (  0d 0' 0.01"E, 24d26'49.74"N)
Band 1 Block=256x256 Type=Byte, ColorInterp=Red
  Overviews: 4194304x4194304, ..., 256x256
  Mask Flags: PER_DATASET ALPHA
  Overviews of mask band: Overviews: 4194304x4194304, ..., 256x256
Band 2 Block=256x256 Type=Byte, ColorInterp=Green
  Overviews: 4194304x4194304, ..., 256x256
  Mask Flags: PER_DATASET ALPHA
  Overviews of mask band: Overviews: 4194304x4194304, ..., 256x256
Band 3 Block=256x256 Type=Byte, ColorInterp=Blue
  Overviews: 4194304x4194304, ..., 256x256
  Mask Flags: PER_DATASET ALPHA
  Overviews of mask band: Overviews: 4194304x4194304, ..., 256x256
Band 4 Block=256x256 Type=Byte, ColorInterp=Alpha
  Overviews: 4194304x4194304, ..., 256x256
```

6.119.6 See Also

- Documentation of Planet Mosaics API
- API Authentication
- *Vector PLScenes / Planet Scenes API driver*

6.120 PMTiles

Added in version 3.14.

Driver short name

PMTiles

Driver built-in by default

This driver is built-in by default

This driver supports reading and writing PMTiles datasets containing raster tiles, encoded in PNG, JPEG or WEBP format.

PMTiles is a single-file archive format for tiled data. A PMTiles archive can be hosted on a commodity storage platform such as S3, and enables low-cost, zero-maintenance map applications that are "serverless" - free of a custom tile backend or third party provider.

This driver is compatible with all GDAL *network-based virtual file systems*

The SRS is always the Pseudo-Mercator (a.k.a Google Mercator) projection, EPSG:3857.

The supported datasets must contain a JSON metadata document following the MBTiles specification.

The driver also supports vector layers. See *PMTiles vector driver*

The driver requires at runtime the availability of the *GTI -- GDAL Raster Tile Index* and *GPKG -- GeoPackage vector* drivers for read support, and *MBTiles* for creation support. For both read and creation, it also requires the *PNG -- Portable Network Graphics*, *JPEG -- JPEG JFIF File Format* or *WEBP - WEBP* driver depending on the tile format of the dataset.

6.120.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.120.2 Opening options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are available:

- **ZOOM_LEVEL=<integer>:** Zoom level to use. Must be between the minimum and maximum zoom levels reported in the metadata. Defaults to the maximum zoom level available in the dataset.

6.120.3 Creation issues

The driver supports conversion from another GDAL supported raster format into PMTiles.

Tiles are generated with WebMercator (EPSG:3857) projection.

Internally the source dataset is converted into a temporary MBTiles dataset, hence both drivers supports the same raster creation options.

The driver implements also a direct translation mode when using **`gdal raster convert`** with a MBTiles raster dataset as input and a PMTiles output dataset, without any creation option: `gdal raster convert in.mbtile out.pmtiles`. In that mode, existing PNG/JPEG/WEBP tiles from the MBTiles files are used as such, contrary to the general writing mode that may involve reprojection and resampling.

6.120.4 Creation options

Dataset creation options can be specified in command-line tools using the syntax `-dsco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following dataset creation options are supported:

- **NAME=value:** Tileset name. Defaults to the basename of the output file/directory. Used to fill metadata records.
- **DESCRIPTION=value:** A description of the tileset. Used to fill metadata records.
- **TYPE=[overlay/baselayer]:** Layer type. Used to fill metadata records.
- **VERSION=value:** Defaults to `1.1`. The version of the tileset, as a dotted notation number, used to set the 'version' metadata item.
- **BLOCKSIZE=<integer>:** Defaults to `256`. Block/tile size in width and height in pixels. Maximum supported size is 4096.
- **TILE_FORMAT=[PNG/PNG8/JPEG/WEBP]:** Defaults to `PNG`. Format used to store tiles. See *Tile formats*.
- **QUALITY=1-100:** Defaults to `75`. Quality setting for JPEG/WEBP compression.
- **ZLEVEL=1-9:** Defaults to `6`. DEFLATE compression level for PNG tiles.
- **DITHER=[YES/NO]:** Defaults to `NO`. Whether to use Floyd-Steinberg dithering (for *`TILE_FORMAT=PNG8`*).
- **ZOOM_LEVEL_STRATEGY=[AUTO/LOWER/UPPER]:** Defaults to `AUTO`. Strategy to determine zoom level. LOWER will select the zoom level immediately below the theoretical computed non-integral zoom level, leading to subsampling. On the contrary, UPPER will select the immediately above zoom level, leading to oversampling. Defaults to AUTO which selects the closest zoom level.
- **RESAMPLING=[NEAREST/BILINEAR/CUBIC/CUBICSPLINE/LANCZOS/MODE/AVERAGE]:** Defaults to `BILINEAR`. Resampling algorithm.
- **WRITE_BOUNDS=[YES/NO]:** Defaults to `YES`. Whether to write the bounds 'metadata' item.

- **ELEVATION_TYPE=['']/'terrain-rgb']:** Defaults to `''`. Type of elevation encoding, suitable for direct generation of Mapbox Terrain-RGB tilesets. See Mapbox Terrain-RGB v1

6.120.5 /vsipmtiles/ virtual file system

See */vsipmtiles/ virtual file system*.

6.120.6 Examples

6.120.6.1 Example 1: Simple translation of a GeoTIFF into PMTiles using default PNG tile format

```
gdal raster convert in.tif out.pmtiles
```

6.120.7 See Also

- *PMTiles vector driver*
- PMTiles specification
- *MBTiles driver*

6.121 PNG -- Portable Network Graphics

Driver short name

PNG

Driver built-in by default

internal libpng provided

GDAL includes support for reading, and creating .png files. Greyscale, pseudo-colored, Paletted, RGB and RGBA PNG files are supported as well as precisions of eight and sixteen bits per sample.

The GDAL PNG Driver is built using the libpng library. Also note that the GeoTIFF driver supports tiled TIFF with DEFLATE compressed tiles, which is the same compression algorithm that PNG at its core uses.

PNG files are linearly compressed, so random reading of large PNG files can be very inefficient (resulting in many restarts of decompression from the start of the file). The maximum dimension of a PNG file that can be created by GDAL is set to 1,000,000x1,000,000 pixels by libpng.

Text chunks are translated into metadata, typically with multiple lines per item. *WLD -- ESRI World File* with the extensions of .pgw, .pngw or .wld will be read. Single transparency values in greyscale files will be recognised as a nodata value in GDAL. Transparent index in paletted images are preserved when the color table is read.

PNG files can be created with a type of PNG, using the CreateCopy() method, requiring a prototype to read from. Writing includes support for the various image types, and will preserve transparency/nodata values. Georeferencing .wld files are written if creation option `WORLDFILE` is set. All pixel types other than 16bit unsigned will be written as eight bit.

XMP metadata can be extracted from the file, and will be stored as XML raw content in the xml:XMP metadata domain.

6.121.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.121.2 Transparency color

Added in version 3.12.

On reading, transparency color`(or index) is reported in the `BACKGROUND_COLOR` dataset metadata item.

On writing, the `BACKGROUND_COLOR` dataset metadata item is used, when present, to set the PNG background color. For RGB/RGBA, the 3 components must be comma separated: `<red>,<green>,<blue>`.

6.121.3 Color Profile Metadata

GDAL can deal with the following color profile metadata in the COLOR_PROFILE domain:

- SOURCE_ICC_PROFILE (Base64 encoded ICC profile embedded in file. If available, other tags are ignored.)
- SOURCE_ICC_PROFILE_NAME : ICC profile name. sRGB is recognized as a special value.
- SOURCE_PRIMARIES_RED (xyY in "x,y,1" format for red primary.)
- SOURCE_PRIMARIES_GREEN (xyY in "x,y,1" format for green primary)
- SOURCE_PRIMARIES_BLUE (xyY in "x,y,1" format for blue primary)
- SOURCE_WHITEPOINT (xyY in "x,y,1" format for whitepoint)
- PNG_GAMMA

Note that these metadata properties can only be used on the original raw pixel data. If automatic conversion to RGB has been done, the color profile information cannot be used.

All these metadata tags can be used as creation options. Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following creation options are available:

- **WORLDFILE=[YES/NO]:** Defaults to `NO`. Force the generation of an associated ESRI world file (with the extension .wld). See *World Files* section for details.
- **ZLEVEL=n=[0-9]:** Defaults to `6`. Set the amount of time to spend on compression. A value of 0 is fast but does no compression, and a value of 9 is slow but does the best compression.
- **TITLE=value:** Title, written in a TEXT or iTXt chunk
- **DESCRIPTION=value:** Description, written in a TEXT or iTXt chunk

- **COPYRIGHT=value:** Copyright, written in a TEXT or iTXt chunk
- **COMMENT=value:** Comment, written in a TEXT or iTXt chunk
- **WRITE_METADATA_AS_TEXT=[YES/NO]:** Whether to write source dataset metadata in TEXT chunks
- **NBITS=[1/2/4]:** Force number of output bits

NOTE: Implemented as pngdataset.cpp.

PNG support is implemented based on the libpng reference library. More information is available at http://www.libpng.org/pub/png.

6.122 PNM -- Netpbm (.pgm, .ppm)

Driver short name

PNM

Driver built-in by default

This driver is built-in by default

GDAL includes support for reading, and creating .pgm (greyscale), and .ppm (RGB color) files compatible with the Netpbm tools. Only the binary (raw) formats are supported.

Netpbm files can be created with a type of PNM.

6.122.1 Creation Options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following creation option is available:

- **MAXVAL=<n>:** Force setting the maximum color value to **n** in the output PNM file. May be useful if you planning to use the output files with software which is not liberal to this value.

NOTE: Implemented as pnmdataset.cpp.

6.122.2 Driver capabilities

Supports Create()

This driver supports the *`GDALDriver::Create()`* operation

Supports CreateCopy()

This driver supports the *`GDALDriver::CreateCopy()`* operation

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.123 PostGISRaster -- PostGIS Raster driver

Driver short name

PostGISRaster

Build dependencies

PostgreSQL library

PostGIS Raster (previously known as WKT Raster) is the project that provides raster support on PostGIS. Since September 26st, 2010, is an official part of PostGIS 2.0+.

This driver was started during the Google Summer of Code 2009, and significantly improved since then.

Currently, the driver provides read-only support to PostGIS Raster data sources.

6.123.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Georeferencing

This driver supports georeferencing

6.123.2 Connecting to a database

To connect to a PostGIS Raster datasource, use a connection string specifying the database name, with additional parameters as necessary

```
PG:"[host=''] [port=''] [dbname='' [user=''] [password=''] [schema=''] [table=''] [column=''] [where=''] [mode=''] [outdb_resolution=''] "
```

Note that the string, up to the part starting with "table=" is a libpq-style connection string. That means that you can leave out unnecessary fields (like password, in some cases).

- **schema** - name of PostgreSQL schema where requested raster table is stored.
- **table** - name of PostGIS Raster table. The table was created by the raster loader (eg. raster2pgsql utility).
- **column** - name of raster column in raster table
- **where** - option is used to filter the results of the raster table. Any SQL-WHERE expression is valid.

- **mode** - option is used to know the expected arrangement of the raster table. There are 2 possible values
  - **mode=1** - ONE_RASTER_PER_ROW mode. In this case, a raster table is considered as a bunch of different raster files. This mode is intended for raster tables storing different raster files. It's the default mode if you don't provide this field in connection string.
  - **mode=2** - ONE_RASTER_PER_TABLE mode. In this case, a raster table is considered as a unique raster file, even if the table has more than one row. This mode is intended for reading tiled rasters from database.
- **outdb_resolution** - Option to specify how out-database rasters should be resolved. Default is server_side.
  - **server_side**: The outDB raster will be fetched by the PostgreSQL server. This implies that outdb rasters are enabled on the server.
  - **client_side**: The outDB raster filenames will be returned to the GDAL PostGISRaster client, which will open it on the client side. This implies that the filename stored on te server can be accessed by the client.
  - **client_side_if_possible**: The outDB raster filenames will be returned to the GDAL PostGISRaster client, which will check if it can access them. If it can, that's equivalent to client_side. Otherwise that's equivalent to server_side. Note that this mode involves extra queries to the server.

6.123.2.1 Additional notes

If a table stores a tiled raster and you execute the driver with mode=1, each image tile will be considered as a different image, and will be reported as a subdataset. There are use cases the driver can't still work with. For example: non-regular blocked rasters. That cases are detected and an error is raised. Anyway, as I've said, the driver is under development, and will work with more raster arrangements ASAP.

There's an additional working mode. If you don't provide a table name, the driver will look for existing raster tables in all allowed database' schemas, and will report each table as a subdataset.

You must use this connection string's format in all the gdal tools, like gdalinfo, gdal_translate, gdalwarp, etc.

6.123.2.2 Performance hints

To get the maximum performance from the driver, it is best to load the raster in PostGIS raster with the following characteristics:

- tiled: -t switch of raster2pgsql
- with overview: -l 2,4,8,... switch of raster2pgsql
- with a GIST spatial index on the raster column: -I switch of raster2pgsql
- with constraints registered: -C switch of raster2pgsql

6.123.3 Examples

To get a summary about your raster via GDAL use gdalinfo:

```
gdalinfo  "PG:host=localhost port=5432 dbname='mydb' user='postgres' password='secret'␣
↪schema='public' table=mytable"
```

For more examples, check the PostGIS Raster FAQ section: Can I export my PostGIS Raster data to other raster formats?

6.123.4 Credits

The driver developers

- Jorge Arévalo (jorgearevalo at libregis.org)
- David Zwarg (dzwarg at azavea.com)

- Even Rouault (even.rouault at spatialys.com)

6.123.5 See Also

- GDAL PostGISRaster driver Wiki
- PostGIS Raster documentation

6.124 PHOTOMOD Raster File

Driver short name

PRF

Driver built-in by default

This driver is built-in by default

PRF or MegaTIFF is an internal format of PHOTOMOD software for storing large images.

This format was developed to store images larger than 4 GB. As a basis for storing data used TIFF or JPEG2000 format. Raster is split into fragments (tiles) such that each fragment does not exceeded a predefined size (e.g., less than 1 GB). An overview file also added to process raster data on a small scales.

PRF files has two variations: 'prf' for imagery data and 'x-dem' for elevation data. Files can be georeferenced, but projection information can be stored only in external files (*.prj).

Image format has the following structure:

- the header XML file 'image_name.prf'/'image_name.x-dem'
- folder 'image_name' with raster subtiles
- files *.tif/*.jp2/*.demtif inside folder 'image_name', containing raster fragments and the overview image

The driver support the data type among Byte, UInt16, UInt32, Float32 or Float64.

6.124.1 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.124.2 See Also

- Racurs company home page
- PHOTOMOD Lite home page

6.125 RasterLite2 - Rasters in SQLite DB

Driver short name

SQLite

Note

The above short name is not a typo. The RasterLite2 functionality is part of the *SQLite / Spatialite RDBMS* driver.

Build dependencies

libsqlite3, librasterlite2, libspatialite

The SQLite driver allows reading and writing SQLite databases containing RasterLite2 coverages.

Those databases can be produced by the utilities of the RasterLite2 distribution, such as rl2tools.
The driver supports reading grayscale, paletted, RGB, multispectral images stored as tiles in the many compressed formats supported by libRasterLite2. The driver also supports reading overviews/pyramids, spatial reference system and spatial extent.

GDAL/OGR must be compiled with sqlite support and against librasterlite2 and libspatialite.

The driver is implemented a unified SQLite / SpatiaLite / RasterLite2 vector and raster capable driver.

6.125.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.125.2 Opening syntax

A RasterLite2 filename can be specified as the connection string. If the file contains a single RasterLite2 coverage, this one will be exposed as the GDAL dataset. If the file contains multiple coverages, each one will be exposed as a subdataset with the syntax RASTERLITE2:filename:coverage_name. See the basic concepts of RasterLite2.

If a coverage is made of several sections, they will be listed as subdatasets of the coverage dataset, so as to be accessed individually. By default, they will be exposed as a unified dataset. The syntax of section-based dataset is RASTERLITE2:filename:coverage_name:section_id:section_name.

6.125.3 Creation

The driver supports creating new databases from scratch, adding new coverages to an existing database and adding sections to an existing coverage.

6.125.4 Creation options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following creation options are supported:

- **APPEND_SUBDATASET=[YES/NO]:** Defaults to `NO`. Whether to add the raster to the existing file. If set to YES, COVERAGE must be specified. Default is NO (ie overwrite existing file)
- **COVERAGE=value:** Coverage name. If not specified, the basename of the output file is used.
- **SECTION=value:** Section name. If not specified, the basename of the output file is used.
- **COMPRESS=[NONE/DEFLATE/LZMA/PNG/CCITTFAX4/JPEG/WEBP/CHARS/JPEG2000]:** Defaults to `NONE`. Compression method. See the information about supported codecs. Note that some codecs may not be available depending on how librasterlite2 has been built.
- **QUALITY=[0-100]:** Image quality for JPEG, WEBP and JPEG2000 compressions. Exact meaning depends on the compression method. For WEBP and JPEG2000, the value 100 triggers the use of their lossless variants.
- **PIXEL_TYPE=[MONOCHROME/PALETTE/GRAYSCALE/RGB/MULTIBAND/DATAGRID]:** Raster pixel type. Determines the photometric interpretation. See the information about supported pixel types. The driver will automatically determine an appropriate pixel type given the band characteristics.
- **BLOCKXSIZE=<int>:** Defaults to `512`. Block width.
- **BLOCKYSIZE=<int>:** Defaults to `512`. Block height.
- **NBITS=[1/2/4]:** Force bit width. This will be by default gotten from the NBITS metadata item in the IMAGE_STRUCTURE metadata domain of the source raster band.
- **PYRAMIDIZE=[YES/NO]:** Defaults to `NO`. Whether to build automatically build relevant pyramids/overviews. Pyramids can be built with the BuildOverviews() / gdaladdo.

6.125.5 Examples

- Reading a RasterLite2 database with a single coverage:

```
gdalinfo my.rl2
```

- Listing the subdatasets corresponding to the coverages of a RasterLite2 database with several coverages:

```
gdalinfo multiple_coverages.rl2
```

- Reading a subdataset corresponding to a coverage:

```
gdalinfo RASTERLITE2:multiple_coverages.rl2:my_coverage
```

- Creating a RasterLite2 dataset from a grayscale image:

```
gdal_translate -f SQLite byte.tif byte.rl2
```

- Creating a RasterLite2 dataset from a RGB image, and using JPEG compression:

```
gdal_translate -f SQLite rgb.tif rgb.rl2 -co COMPRESS=JPEG
```

- Adding a RasterLite2 coverage to an existing SpatiaLite/RasterLite2 database:

```
gdal_translate -f SQLite rgb.tif rgb.rl2 -co APPEND_SUBDATASET=YES -co COVERAGE=rgb
```

- Adding pyramids to a coverage:

```
gdaladdo rgb.rl2 2 4 8 16
```

6.125.6 See Also

- Rasterlite2 home page
- *OGR SQLite driver*

6.126 RCM -- RADARSAT Constellation Mission Product

Added in version 3.11.

Driver short name

RCM

Driver built-in by default

This driver is built-in by default

This driver will read RADARSAT Constellation Mission polarimetric products.

The RADARSAT Constellation Mission XML products are distributed with a primary XML file called product.xml, and a set of supporting XML data files with the actual imagery stored in TIFF files. The RCM driver will be used if the product.xml or the containing directory is selected, and it can treat all the imagery as one consistent dataset.

The RCM driver also reads geolocation tiepoints from the product.xml file and represents them as GCPs on the dataset.

6.126.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.126.2 Data Calibration

If you wish to have GDAL apply a particular calibration LUT to the data when you open it, you have to open the appropriate subdatasets. The following subdatasets exist within the SUBDATASET domain for RCM products:

- uncalibrated: open with `RCM_CALIB:UNCALIB:` prepended to filename
- $beta_0$: open with `RCM_CALIB:BETA0:` prepended to filename
- $sigma_0$: open with `RCM_CALIB:SIGMA0:` prepended to filename
- Gamma: open with `RCM_CALIB:GAMMA:` prepended to filename

Note that geocoded (GCC/GCD) products do not have this functionality available. Also be aware that the LUTs must be in the product directory where specified in the product.xml, otherwise loading the product with the calibration LUT applied will fail.

One caveat worth noting is that the RCM driver will supply the calibrated data as GDT_Float32 or GDT_CFloat32 depending on the type of calibration selected. The uncalibrated data is provided as GDT_Int16/GDT_UInt8/GDT_CInt16, also depending on the type of product selected.

6.126.3 See Also

- RADARSAT Constellation Mission Product Specification RCM-SP-52-9092

6.127 RIK -- Swedish Grid Maps

Driver short name

RIK

Build dependencies

(internal zlib is used if necessary)

Supported by GDAL for read access. This format is used in maps issued by the swedish organization Lantmäteriet. Supports versions 1, 2 and 3 of the RIK format, but only 8 bits per pixel.

This driver is based on the work done in the TRikPanel project.

NOTE: Implemented as rikdataset.cpp.

6.127.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.128 RMF -- Raster Matrix Format

Driver short name

RMF

Driver built-in by default

This driver is built-in by default

RMF is a simple tiled raster format used in the GIS "Integration" and "Panorama" GIS. The format itself has very poor capabilities.

There are two flavors of RMF called MTW and RSW. MTW supports 16-bit integer and 32/64-bit floating point data in a single channel and aimed to store DEM data. RSW is a general purpose raster. It supports single channel colormapped or three channel RGB images. Only 8-bit data can be stored in RSW. Simple georeferencing can be provided for both image types.

6.128.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.128.2 Metadata

- **ELEVATION_MINIMUM**: Minimum elevation value (MTW only).
- **ELEVATION_MAXIMUM**: Maximum elevation value (MTW only).
- **ELEVATION_UNITS**: Name of the units for raster values (MTW only). Can be "m" (meters), "cm" (centimeters), "dm" (decimeters), "mm" (millimeters).
- **ELEVATION_TYPE**: Could be either 0 (absolute elevation) or 1 (total elevation). MTW only.

6.128.3 Open Options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are supported:

- **RMF_SET_VERTCS=ON:** Set to ON, the layers spatial reference will include vertical coordinate system description if exist. This feature can be enabled via config option with same name.

6.128.4 Creation Options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following creation options are supported:

- **MTW=ON:** Force the generation of MTW matrix (RSW will be created by default).
- **BLOCKXSIZE=<int>:** Defaults to `256`. Sets tile width.
- **BLOCKYSIZE=<int>:** Defaults to `256`. Set tile height.
- **RMFHUGE=[NO/YES/IF_SAFER]:** Defaults to `NO`. Creation of huge RMF file (Supported by GIS Panorama since v11).
- **COMPRESS=[NONE/LZW/JPEG/RMF_DEM]:** Defaults to `NONE`. Compression type. Note: JPEG compression supported only with RGB (3-band) Byte datasets. RMF_DEM compression supported only with Int32 one channel MTW datasets.
- **JPEG_QUALITY=[1-100]:** Defaults to `75`. JPEG quality 1-100.
- **NUM_THREADS=[<int>/ALL_CPUS]:** Enable multi-threaded compression by specifying the number of worker threads. Default is compression in the main thread.

6.128.5 See Also:

- Implemented as rmfdataset.cpp.
- "Panorama" GIS homepage

6.129 ROI_PAC -- ROI_PAC

Driver short name

ROI_PAC

Driver built-in by default

This driver is built-in by default

Driver for the image formats used in the JPL's ROI_PAC project (https://aws.roipac.org/). All image type are supported excepted .raw images.

Metadata are stored in the ROI_PAC domain.

Georeferencing is supported, but expect problems when using the UTM projection, as ROI_PAC format do not store any hemisphere field.

When creating files, you have to be able to specify the right data type corresponding to the file type (slc, int, etc), else the driver will output an error.

NOTE: Implemented as roipacdataset.cpp.

6.129.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.130 RPFTOC -- Raster Product Format/RPF (a.toc)

Driver short name

RPFTOC

Driver built-in by default

This driver is built-in by default

This is a reader for RPF products, like CADRG or CIB, that uses the table of content file - A.TOC - from a RPF exchange, and exposes it as a virtual dataset whose coverage is the set of frames contained in the table of content.

Starting with GDAL 3.13, the A.TOC file can also be generated using the `gdal driver rpftoc create` program from existing CADRG frames.

The driver will report a different subdataset for each subdataset found in the A.TOC file.

Result of a gdalinfo on a A.TOC file.

```
Subdatasets:
  SUBDATASET_1_NAME=NITF_TOC_ENTRY:CADRG_GNC_5M_1_1:GNCJNCN/rpf/a.toc
  SUBDATASET_1_DESC=CADRG:GNC:Global Navigation Chart:5M:1:1
```

(continues on next page)

(continued from previous page)

```
[...]
  SUBDATASET_5_NAME=NITF_TOC_ENTRY:CADRG_GNC_5M_7_5:GNCJNCN/rpf/a.toc
  SUBDATASET_5_DESC=CADRG:GNC:Global Navigation Chart:5M:7:5
  SUBDATASET_6_NAME=NITF_TOC_ENTRY:CADRG_JNC_2M_1_6:GNCJNCN/rpf/a.toc
  SUBDATASET_6_DESC=CADRG:JNC:Jet Navigation Chart:2M:1:6
[...]
  SUBDATASET_13_NAME=NITF_TOC_ENTRY:CADRG_JNC_2M_8_13:GNCJNCN/rpf/a.toc
  SUBDATASET_13_DESC=CADRG:JNC:Jet Navigation Chart:2M:8:13
```

It is possible to build external overviews for a subdataset. The overview for the first subdataset will be named A.TOC.1.ovr for example, for the second dataset it will be A.TOC.2.ovr, etc. Note that you must re-open the subdataset with the same setting of *RPFTOC_FORCE_RGBA* as the one you have used when you have created it. Do not use any method other than NEAREST resampling when building overviews on a paletted subdataset (*RPFTOC_FORCE_RGBA* unset)

A gdalinfo on one of this subdataset will return the various NITF metadata, as well as the list of the NITF tiles of the subdataset.

See Also:

- MIL-PRF-89038 : specification of RPF, CADRG, CIB products

NOTE: Implemented as rpftocdataset.cpp

6.130.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.130.2 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). This driver supports the following open options:

- **FORCE_RGBA=[YES/NO]:** (GDAL >= 3.13) Defaults to `NO`. In some situations, *NITF -- National Imagery Transmission Format (also CIB, CADRG, ECRG, HRE)* tiles inside a subdataset don't share the same palettes. The RPFTOC driver will do its best to remap palettes to the reported palette by gdalinfo (which is the palette of the first tile of the subdataset). If it doesn't produce the desired result, you can set this option to YES to get a RGBA dataset, instead of a paletted one. Equivalent to setting the *RPFTOC_FORCE_RGBA* configuration option.

6.130.3 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). This paragraph lists the configuration options that can be set to alter the default behavior of the RPFTC driver.

- **RPFTOC_FORCE_RGBA=[YES/NO]:** Defaults to `NO`. Equivalent to setting the *FORCE_RGBA* open option.

6.130.4 Creation of a A.TOC file from existing CADRG frames

Added in version 3.13.

6.130.4.1 Description

gdal driver rpftoc create can be used to create a A.TOC file from existing CADRG frames. The value of the `input` argument should be the path to the `RPF` directory under which the CADRG frames are found. The `A.TOC` file will be written in that directory.

6.130.4.2 Synopsis

```
Usage: gdal driver rpftoc create [OPTIONS] <INPUT> [<OUTPUT>]

Create a A.TOC index from CADRG frames.

Positional arguments:
  -i, --input <INPUT>                Input directory [required]
  -o, --output <OUTPUT>              Output filename

Common Options:
  -h, --help                         Display help message and exit
  --json-usage                       Display usage as JSON document and exit
  --config <KEY>=<VALUE>             Configuration option [may be repeated]

Options:
  --scale <SCALE>                    (Reciprocal) scale (e.g. 1000000)
  --producer-id <PRODUCER-ID>        Producer (short) identification
  --producer-name <PRODUCER-NAME>    Producer name
  --country-code <COUNTRY-CODE>      ISO country code for security
  --classification <CLASSIFICATION>  Index classification. CLASSIFICATION=U|R|C|S|T␣
↪(default: U)
```

6.130.4.3 Program-Specific Options

--scale `<SCALE>`

(Reciprocal) scale (e.g. 1000000). If not specified, it will be guessed from the content of CADRG frames (except for those where this cannot be inferred automatically)

--producer-id `<PRODUCER-ID>`

Producer (short) identification. Up to 5 characters.

--producer-name `<PRODUCER-NAME>`

Producer name. Up to 10 characters.

--contry-code `<CONTRY-CODE>`

Two letter ISO country code for security classification

--classification `U|R|C|S|T`

Index classification. Defaults to U (Unclassified)

6.130.4.4 Examples

```
gdal driver rpftoc create /path/to/RPF --producer-id NIMA
```

6.131 RRASTER -- R Raster

Driver short name

RRASTER

Driver built-in by default

This driver is built-in by default

This is a read-only reader for the datasets handled by the R Raster package. Those datasets are made of a .grd file, which is a text header file, and a .gri binary file containing the raster data itself. The .grd is the file opened by GDAL. The driver will read ratvalues as RAT or color tables. Layer names will be assigned to GDAL band description. The 'creator' and 'created' attributes of the '[general]' section will be assigned to the GDAL 'CREATOR' and 'CREATED' dataset metadata items.

The driver has write capabilities. Color tables or RAT will be written. The 'CREATOR' and 'CREATED' dataset metadata items will be written as the 'creator' and 'created' attributes of the '[general]' section. Band description will be written as the 'layername' attribute of the '[description]' section.

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following creation options are supported:

- **INTERLEAVE=[BIP/BIL/BSQ]:** Defaults to `BIL`. Respectively band interleaved by pixel, band interleaved by line, band sequential. Starting with GDAL 3.5, when copying from a source dataset with multiple bands which advertises a INTERLEAVE metadata item, if the INTERLEAVE creation option is not specified, the source dataset INTERLEAVE will be automatically taken into account. See *Multiband pixel organization (INTERLEAVE metadata item)* for more details.
- **PIXELTYPE=SIGNEDBYTE:** To write Byte bands as signed byte instead of unsigned byte. Starting with GDAL 3.7, this option is deprecated and Int8 should rather be used.

6.131.1 Driver capabilities

Supports CreateCopy()

This driver supports the *`GDALDriver::CreateCopy()`* operation

Supports Create()

This driver supports the *`GDALDriver::Create()`* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.131.2 See Also

- Description of the "rasterfile" format

6.132 RS2 -- RadarSat 2 XML Product

Driver short name

RS2

Driver built-in by default

This driver is built-in by default

This driver will read some RadarSat 2 XML polarimetric products. In particular complex products, and 16bit magnitude detected products.

The RadarSat 2 XML products are distributed with a primary XML file called product.xml, and a set of supporting XML data files with the actual imagery stored in TIFF files. The RS2 driver will be used if the product.xml or the containing directory is selected, and it can treat all the imagery as one consistent dataset.

The complex products use "32bit void typed" TIFF files which are not meaningfully readable normally. The RS2 driver takes care of converting this into useful CInt16 format internally.

The RS2 driver also reads geolocation tiepoints from the product.xml file and represents them as GCPs on the dataset.

It is very likely that RadarSat International will be distributing other sorts of datasets in this format; however, at this time this driver only supports specific RadarSat 2 polarimetric products. All other will be ignored, or result in various runtime errors. It is hoped that this driver can be generalized when other product samples become available.

6.132.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.132.2 Data Calibration

If you wish to have GDAL apply a particular calibration LUT to the data when you open it, you have to open the appropriate subdatasets. The following subdatasets exist within the SUBDATASET domain for RS2 products:

- uncalibrated - open with RADARSAT_2_CALIB:UNCALIB: prepended to filename
- $beta_0$ - open with RADARSAT_2_CALIB:BETA0: prepended to filename
- $sigma_0$ - open with RADARSAT_2_CALIB:SIGMA0: prepended to filename
- gamma - open with RADARSAT_2_CALIB:GAMMA: prepended to filename

Note that geocoded (SPG/SSG) products do not have this functionality available. Also be aware that the LUTs must be in the product directory where specified in the product.xml, otherwise loading the product with the calibration LUT applied will fail.

One caveat worth noting is that the RADARSAT-2 driver will supply the calibrated data as GDT_Float32 or GDT_CFloat32 depending on the type of calibration selected. The uncalibrated data is provided as GDT_Int16/GDT_UInt8/GDT_CInt16, also depending on the type of product selected.

6.132.3 See Also

- RadarSat document RN-RP-51-27.

6.133 S102 -- S-102 Bathymetric Surface Product

Driver short name

S102

Build dependencies

libhdf5

Added in version 3.8.

This driver provides support for bathymetry data in the S-102 format, which is a specific product profile in an HDF5 file.

S-102 files have two image bands representing depth (band 1), uncertainty (band 2) values for each cell in a raster grid area.

Note that positive values of depth mean values *below* the reference surface of the vertical datum. The `DEPTH_OR_ELEVATION` open option can be set to `ELEVATION` to expose depth values as elevation values, by negating their sign (i.e. positive values of elevation mean values above the reference surface)

Georeferencing is reported.

Nodata, minimum and maximum values for each band are also reported.

For reading, supported versions of the specification are S-102 v2.1, v2.2 and v3.0.

Since GDAL 3.12, multiple feature instance groups per dataset (to encode grids using different vertical datums) are supported. In that case, each feature instance group is exposed as a GDAL subdataset, whose name is of the form `S102:"{filename.h5}":BathymetricCoverage.{XX}`.

Write support for S-102 v3.0 has been added in GDAL 3.13

6.133.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.133.2 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are supported:

- **DEPTH_OR_ELEVATION=[DEPTH/ELEVATION]:** Defaults to `DEPTH`. Whether to report depth or elevation. Positive values of depth mean values *below* the reference surface of the vertical datum. Positive values of elevation mean values *above* the reference surface of the vertical datum (which is the convention used by the *BAG driver*)
- **NORTH_UP=[YES/NO]:** Defaults to `YES`. Whether the top line of the dataset should be the northern-most one. This is the default behavior of most GDAL formats, but the native organization of the data in S-102 products is to have the first line of the grid being the southern-most one. This native organization can be exposed by the driver by setting this option to NO (in which case the 6th term of the geotransform matrix will be positive)

6.133.3 Spatial metadata support

Starting with GDAL 3.9, GDAL can handle QualityOfSurvey (or QualityOfBathymetryCoverage in S102 v3.0) spatial metadata.

When such spatial metadata is present, the subdataset list will include a name of the form `S102:"{filename}":QualityOfSurvey` (or `S102:"{filename}":QualityOfBathymetryCoverage` in S102 v3.0)

The `/QualityOfSurvey/featureAttributeTable` (`/QualityOfBathymetryCoverage/featureAttributeTable` in S102 v3.0) dataset is exposed as a GDAL Raster Attribute Table associated to the GDAL raster band. The pixel values of the raster match the `id` column of the Raster Attribute Table.

6.133.4 Write support

Added in version 3.13.

Creation of a S-102 v3.0 dataset from another existing GDAL supported dataset is possible using the *GDALDriver::CreateCopy()* function, or utilities like *gdal_translate* or *gdal raster convert*. The input dataset must have one or two bands. The first band must represent depths (positive values are down below the vertical datum surface) in meters, and the optional second band must contain the uncertainty value in meters. If the first band has a description "elevation" (typically for BAG datasets), elevations will be automatically converted to depths by negating the sign of source values.

The input dataset must be in the following CRS:

- WGS 84 longitude/latitude (EPSG:4326)
- Any of the 60 north/south UTM projected CRS over WGS 84 (EPSG:32601 to 32660, or 32701 to 32760)
- "WGS 84 / UPS North (E,N)" (EPSG:5041)
- "WGS 84 / UPS South (E,N)" (EPSG:5042)

If several vertical datums are needed, the *APPEND_SUBDATASET* creation option can be set to `YES` to add an extra feature instance group ("BathymetricCoverage.XX") to an existing S-102 dataset.

The following creation options are available:

- **VERTICAL_DATUM=<string>:** Vertical datum. This is a required creation option. Possible values are either a S100 vertical datum numeric code in the 1 to 30 range, or value 44. Or their string meaning or abbreviation among the following list:
  - 1: `meanLowWaterSprings` / `MLWS`
  - 2: `meanLowerLowWaterSprings`
  - 3: `meanSeaLevel` / `MSL`
  - 4: `lowestLowWater`
  - 5: `meanLowWater` / `MLW`
  - 6: `lowestLowWaterSprings`
  - 7: `approximateMeanLowWaterSprings`
  - 8: `indianSpringLowWater`
  - 9: `lowWaterSprings`
  - 10: `approximateLowestAstronomicalTide`
  - 11: `nearlyLowestLowWater`
  - 12: `meanLowerLowWater` / `MLLW`
  - 13: `lowWater` / `LW`
  - 14: `approximateMeanLowWater`
  - 15: `approximateMeanLowerLowWater`
  - 16: `meanHighWater` / `MHW`
  - 17: `meanHighWaterSprings` / `MHWS`
  - 18: `highWater` / `HW`
  - 19: `approximateMeanSeaLevel`
  - 20: `highWaterSprings`
  - 21: `meanHigherHighWater` / `MHHW`
  - 22: `equinoctialSpringLowWater`
  - 23: `lowestAstronomicalTide` / `LAT`
  - 24: `localDatum`
  - 25: `internationalGreatLakesDatum1985`
  - 26: `meanWaterLevel`
  - 27: `lowerLowWaterLargeTide`

  - 28: `higherHighWaterLargeTide`
  - 29: `nearlyHighestHighWater`
  - 30: `highestAstronomicalTide` / `HAT`
  - 44: `balticSeaChartDatum2000`
- **ISSUE_DATE=<string> as <YYYYMMDD>:** If not specified, defaults to the current date.
- **ISSUE_TIME=<string>:** Issue time as <hhmmssZ> or <hhmmss[+-]HHMM>
- **HORIZONTAL_POSITION_UNCERTAINTY=<number>:** Horizontal position uncertainty in meter
- **VERTICAL_UNCERTAINTY=<number>:** Vertical uncertainty in meter
- **QUALITY_DATASET=<string>:** Path to a dataset with the quality of bathymetric coverage (spatial metadata). This must point to a GDAL recognized dataset, with a single band of an integer data type, containing quality codes. The band must be associated with a raster attribute table, with an integer column named `id` for each value in the band pixels, and optional columns among the ones allowed by the S-102 specification:
  - `dataAssessment` (uint8)
  - `featuresDetected.leastDepthOfDetectedFeaturesMeasured` (boolean)
  - `featuresDetected.significantFeaturesDetected` (boolean)
  - `featuresDetected.sizeOfFeaturesDetected` (float32)
  - `featureSizeVar` (float32)
  - `fullSeafloorCoverageAchieved` (boolean)
  - `bathyCoverage` (boolean)
  - `zoneOfConfidence.horizontalPositionUncertainty.uncertaintyFixed` (float32)
  - `zoneOfConfidence.horizontalPositionUncertainty.uncertaintyVariableFactor` (float32)
  - `surveyDateRange.dateStart` (date)
  - `surveyDateRange.dateEnd` (date)
  - `sourceSurveyID` (string)
  - `surveyAuthority` (string)
  - `typeOfBathymetricEstimationUncertainty` (enumeration)
- **COMPRESS=[NONE/DEFLATE]:** Defaults to `DEFLATE`. Compression for elevation and uncertainty grids.
- **ZLEVEL=1-9:** Defaults to `6`. Deflate compression level.
- **BLOCK_SIZE=<integer>:** Chunking size of the HDF5 arrays. Default to 100, or the maximum dimension of the raster if smaller than 100.
- **APPEND_SUBDATASET=[YES/NO]:** Defaults to `NO`. Whether to append the new dataset to an existing S-102 dataset as an extra feature instance group ("BathymetricCoverage.XX")

6.133.5 Validation script

Added in version 3.13.

The Python script validate_s102.py can be used to validate the conformity of a S-102 v3.0 dataset against the specification. It requires the h5py Python module to be installed (typically through "pip install h5py")

Its usage is:

```
$ python validate_s102.py 102TESTXXXX.h5
```

Note that the GDAL S-102 reader is more tolerant that the validation script and can read files with slight non-conformities.

6.133.6 Examples

- Converting a GeoTIFF with depth and uncertainty and another one with quality information to a S-102 dataset

```
$ gdal_translate depth_uncertainty.tif 102TESTXXXX.h5 -of S102 -co VERTICAL_
↪DATUM=MMLW -co QUALITY_DATASET=quality.tif
```

6.133.7 See Also

- Implemented as s102dataset.cpp.
- S-102 Bathymetric Surface Product Specification
- *BAG driver*
- *S-104 driver*
- *S-111 driver*

6.134 S104 -- S-104 Water Level Information for Surface Navigation Product

Driver short name

S104

Build dependencies

libhdf5

Added in version 3.9.

This driver provides support for water level data in the S-104 format, which is a specific product profile in an HDF5 file.

S-104 files have two image bands representing water level height (band 1) and water level trend (band 2) values for each cell in a raster grid area. An optional third band can contain the uncertainty in water level height.

When opening a S-104 file, no raster band is directly available. But a list of subdatasets will be reported, one for each timestamp available in the file.

An actual dataset can be opened through such a subdataset, with a syntax like `S104:"{filename.h5}":Group_001`.

Georeferencing is reported.

Note that the driver currently only supports regularly gridded S104 datasets.

Since GDAL 3.12, multiple feature instance groups per dataset (to encode grids using different vertical datums) are supported. In that case, each feature instance group and timestamp group is exposed as a GDAL subdataset, whose name is of the form `S104:"{filename.h5}":WaterLevel.{XX}:Group_{YYY}`.

Write support for S-104 v2.0 has been added in GDAL 3.13

6.134.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.134.2 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are supported:

- **NORTH_UP=[YES/NO]:** Defaults to `YES`. Whether the top line of the dataset should be the northern-most one. This is the default behavior of most GDAL formats, but the native organization of the data in S-104 products is to have the first line of the grid being the southern-most one. This native organization can be exposed by the driver by setting this option to NO (in which case the 6th term of the geotransform matrix will be positive)

6.134.3 Write support

Added in version 3.13.

Creation of a S-104 v2.0 dataset from another existing GDAL supported dataset is possible using the *GDALDriver::CreateCopy()* function, or utilities like *gdal_translate* or *gdal raster convert*. The input dataset must have two or three bands. The first band must represent height in meters, and the second band must contain the water level trend value with the following codes:

- 0 = nodata
- 1 = decreasing
- 2 = increasing
- 3 = steady

The third band when present must contain the uncertainty of water level heights (in meters).

If several grids are available at different timestamps, they can be provided with the *DATASETS* creation option, possibly with the *DATASETS_TIME_POINT* creation option if the datasets do not have a `timePoint` metadata item.

If several vertical datums are needed, the *APPEND_SUBDATASET* creation option can be set to `YES` to add an extra feature instance group ("WaterLevel.XX") to an existing S-104 dataset.

The following creation options are available:

- **TIME_POINT=<string>:** Timestamp as YYYYMMDDTHHMMSSZ format (required).
- **VERTICAL_DATUM=<string>:** Vertical datum. This is a required creation option. Possible values are either:

- a S100 vertical datum. Expressed as numeric code in the 1 to 30 range, or value 44, 46, 47, 48, 49. Or their string meaning or abbreviation among the following list:
  * 1: `meanLowWaterSprings` / `MLWS`
  * 2: `meanLowerLowWaterSprings`
  * 3: `meanSeaLevel` / `MSL`
  * 4: `lowestLowWater`
  * 5: `meanLowWater` / `MLW`
  * 6: `lowestLowWaterSprings`
  * 7: `approximateMeanLowWaterSprings`
  * 8: `indianSpringLowWater`
  * 9: `lowWaterSprings`
  * 10: `approximateLowestAstronomicalTide`
  * 11: `nearlyLowestLowWater`
  * 12: `meanLowerLowWater` / `MLLW`
  * 13: `lowWater` / `LW`
  * 14: `approximateMeanLowWater`
  * 15: `approximateMeanLowerLowWater`
  * 16: `meanHighWater` / `MHW`
  * 17: `meanHighWaterSprings` / `MHWS`
  * 18: `highWater` / `HW`
  * 19: `approximateMeanSeaLevel`
  * 20: `highWaterSprings`
  * 21: `meanHigherHighWater` / `MHHW`
  * 22: `equinoctialSpringLowWater`
  * 23: `lowestAstronomicalTide` / `LAT`
  * 24: `localDatum`
  * 25: `internationalGreatLakesDatum1985`
  * 26: `meanWaterLevel`
  * 27: `lowerLowWaterLargeTide`
  * 28: `higherHighWaterLargeTide`
  * 29: `nearlyHighestHighWater`
  * 30: `highestAstronomicalTide` / `HAT`
  * 44: `balticSeaChartDatum2000`
  * 46: `internationalGreatLakesDatum2020`
  * 47: `seaFloor`
  * 48: `seaSurface`

    * 49: `hydrographicZero`
  - an EPSG vertical datum code

- **VERTICAL_CS=DEPTH or HEIGHT:** Vertical coordinate system. This is a required creation option. Depth is the down direction relative to the vertical datum surface, with meter unit. Height is the up direction relative to the vertical datum surface, with meter unit.
- **WATER_LEVEL_TREND_THRESHOLD=<number>:** Critical value used to determine steady water level trend. Units are meters/hour (m/hr). This is a required creation option.
- **DATA_DYNAMICITY=[observation/astronomicalPrediction/analysisOrHybrid or hydrodynamicForecast]:** Classification of data according to the relationship between the time of its collection, generation, or calculation of generation parameters, in relation to the time of publication of the dataset. This is a required creation option.
- **DATASETS=<string>:** Comma-separated list of datasets at different timestamps. If each dataset does not have a `timePoint` metadata item, the *`DATASETS_TIME_POINT`* creation option must be specified. The source dataset may or may not be put in the DATASETS creation option.
- **DATASETS_TIME_POINT=<string>:** Comma-separated list of different timestamps. If must have the same number of values as the *`DATASETS`* creation option. Each time point value must be specified as a `YYYYMMDDTHHMMSSZ` timestamp.
- **ISSUE_DATE=<string> as <YYYYMMDD>:** If not specified, defaults to the current date.
- **ISSUE_TIME=<string>:** Issue time as <hhmmssZ> or <hhmmss[+-]HHMM> If not specified, defaults to the current time (in Z timezone).
- **TREND_INTERVAL=<number>:** Interval, in minutes, over which trend at a a particular time is calculated
- **DATASET_DELIVERY_INTERVAL=<string>:** Expected time interval between availability of successive datasets for time-varying data. Must be formatted as `PnYnMnDTnHnMnS` (ISO-8601 duration)
- **TIME_RECORD_INTERVAL=<number>:** Interval in seconds between time records.
- **COMMON_POINT_RULE=[average/low/high/all]:** Defaults to `all`. Procedure used for evaluating the coverage at a position that falls on the boundary or in an area of overlap between geographic objects.
- **UNCERTAINTY=<number>:** Uncertainty of depth values in meter
- **HORIZONTAL_POSITION_UNCERTAINTY=<number>:** Horizontal position uncertainty in meter
- **VERTICAL_UNCERTAINTY=<number>:** Vertical uncertainty in meter
- **TIME_UNCERTAINTY=<number>:** Time uncertainty in second
- **COMPRESS=[NONE/DEFLATE]:** Defaults to `DEFLATE`. Compression for elevation and uncertainty grids.
- **ZLEVEL=1-9:** Defaults to `6`. Deflate compression level.
- **BLOCK_SIZE=<integer>:** Chunking size of the HDF5 arrays. Default to 100, or the maximum dimension of the raster if smaller than 100.
- **APPEND_SUBDATASET=[YES/NO]:** Defaults to `NO`. Whether to append the new dataset to an existing S-104 dataset as an extra feature instance group ("WaterLevel.XX")

6.134.4 Validation script

Added in version 3.13.

The Python script validate_s104.py can be used to validate the conformity of a S-104 v2.0 dataset against the specification. It requires the h5py Python module to be installed (typically through "pip install h5py")

Its usage is:

```
$ python validate_s104.py 104TESTXXXX.h5
```

Note that the GDAL S-104 reader is more tolerant that the validation script and can read files with slight non-conformities.

6.134.5 Examples

- Converting a GeoTIFF with water level height and trend, with the minimum required metadata items:

```
$ gdal_translate water_level_height_and_trend.tif 104TESTXXXX.h5 -of S104 \
    -co TIME_POINT=20251105T012600Z \
    -co VERTICAL_DATUM=MMLW \
    -co VERTICAL_CS=HEIGHT \
    -co WATER_LEVEL_TREND_THRESHOLD=0.2 \
    -co DATA_DYNAMICITY=hydrodynamicForecast
```

6.134.6 See Also

- Implemented as s104dataset.cpp.
- S-104 Bathymetric Surface Product Specification
- *BAG driver*
- *S-102 driver*
- *S-111 driver*

6.135 S111 -- S-111 Surface Currents Product

Driver short name

S111

Build dependencies

libhdf5

Added in version 3.9.

This driver provides support for surface currents in the S-111 format, which is a specific product profile in an HDF5 file.

S-111 files have at a minimum two image bands representing the following values for each cell in a raster grid area:

- surface current speed (band 1), in knots
- surface current direction (band 2), in degree measured from true north clock-wise.

Additional optional bands can be present to represent the speed uncertainty (in knots) and the direction uncertainty (in degree).

When opening a S-111 file, no raster band is directly available. But a list of subdatasets will be reported, one for each timestamp available in the file.

An actual dataset can be opened through such a subdataset, with a syntax like `S111:"{filename.h5}":Group_001`.

Georeferencing is reported.

Note that the driver currently only supports regularly gridded S111 datasets.

Since GDAL 3.12, multiple feature instance groups per dataset are supported. In that case, each feature instance group and timestamp group is exposed as a GDAL subdataset, whose name is of the form `S111:"{filename.h5}":SurfaceCurrent.{XX}:Group_{YYY}`.

Write support for S-111 v2.0 has been added in GDAL 3.13

6.135.1 Driver capabilities

Supports CreateCopy()

This driver supports the *`GDALDriver::CreateCopy()`* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.135.2 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are supported:

- **NORTH_UP=[YES/NO]:** Defaults to `YES`. Whether the top line of the dataset should be the northern-most one. This is the default behavior of most GDAL formats, but the native organization of the data in S-111 products is to have the first line of the grid being the southern-most one. This native organization can be exposed by the driver by setting this option to NO (in which case the 6th term of the geotransform matrix will be positive)

6.135.3 Write support

Added in version 3.13.

Creation of a S-111 v2.0 dataset from another existing GDAL supported dataset is possible using the *`GDALDriver::CreateCopy()`* function, or utilities like *gdal_translate* or *gdal raster convert*. The input dataset must have two or four bands. The first band must contain the surface current speed in knot, and the second band must contain the surface current direction, in degree measured from true north clock-wise, between 0 and 359.9.

The third and fourth bands when present must contain respectively the uncertainty of the speed (in knot) and the uncertainty of the direction (in degree)

If several grids are available at different timestamps, they can be provided with the *`DATASETS`* creation option, possibly with the *`DATASETS_TIME_POINT`* creation option if the datasets do not have a `timePoint` metadata item.

If several feature instances are needed, the *`APPEND_SUBDATASET`* creation option can be set to `YES` to add an extra feature instance group ("SurfaceCurrent.XX") to an existing S-111 dataset.

The following creation options are available:

- **TIME_POINT=<string>:** Timestamp as YYYYMMDDTHHMMSSZ format (required).

- **DEPTH_TYPE=``heightOrDepth`` or ``layerAverage``:** Type of depth (required). When selecting `heightOrDepth`, the interpretation depends on the VERTICAL_CS value.
- **VERTICAL_DATUM=<string>:** Vertical datum. This is a required creation option when `DEPTH_TYPE=heightOrDepth`. Possible values are either:
  - a S100 vertical datum. Expressed as numeric code in the 1 to 30 range, or value 44, 46, 47, 48, 49. Or their string meaning or abbreviation among the following list:
    - 1: `meanLowWaterSprings` / `MLWS`
    - 2: `meanLowerLowWaterSprings`
    - 3: `meanSeaLevel` / `MSL`
    - 4: `lowestLowWater`
    - 5: `meanLowWater` / `MLW`
    - 6: `lowestLowWaterSprings`
    - 7: `approximateMeanLowWaterSprings`
    - 8: `indianSpringLowWater`
    - 9: `lowWaterSprings`
    - 10: `approximateLowestAstronomicalTide`
    - 11: `nearlyLowestLowWater`
    - 12: `meanLowerLowWater` / `MLLW`
    - 13: `lowWater` / `LW`
    - 14: `approximateMeanLowWater`
    - 15: `approximateMeanLowerLowWater`
    - 16: `meanHighWater` / `MHW`
    - 17: `meanHighWaterSprings` / `MHWS`
    - 18: `highWater` / `HW`
    - 19: `approximateMeanSeaLevel`
    - 20: `highWaterSprings`
    - 21: `meanHigherHighWater` / `MHHW`
    - 22: `equinoctialSpringLowWater`
    - 23: `lowestAstronomicalTide` / `LAT`
    - 24: `localDatum`
    - 25: `internationalGreatLakesDatum1985`
    - 26: `meanWaterLevel`
    - 27: `lowerLowWaterLargeTide`
    - 28: `higherHighWaterLargeTide`
    - 29: `nearlyHighestHighWater`
    - 30: `highestAstronomicalTide` / `HAT`
    - 44: `balticSeaChartDatum2000`

    * 46: `internationalGreatLakesDatum2020`
    * 47: `seaFloor`
    * 48: `seaSurface`
    * 49: `hydrographicZero`
  - an EPSG vertical datum code

- **VERTICAL_CS=DEPTH or HEIGHT:** Vertical coordinate system. This is a required creation option when `DEPTH_TYPE=heightOrDepth`. Depth is the down direction relative to the vertical datum surface, with meter unit. Height is the up direction relative to the vertical datum surface, with meter unit.
- **SURFACE_CURRENT_DEPTH=<number>:** Depth/height value or layer thickness (m) This is a required creation option.
- **DATA_DYNAMICITY=[observation/astronomicalPrediction/analysisOrHybrid or hydrodynamicForecast]:** Classification of data according to the relationship between the time of its collection, generation, or calculation of generation parameters, in relation to the time of publication of the dataset. This is a required creation option.
- **DATASETS=<string>:** Comma-separated list of datasets at different timestamps. If each dataset does not have a `timePoint` metadata item, the *`DATASETS_TIME_POINT`* creation option must be specified. The source dataset may or may not be put in the DATASETS creation option.
- **DATASETS_TIME_POINT=<string>:** Comma-separated list of different timestamps. If must have the same number of values as the *`DATASETS`* creation option. Each time point value must be specified as a `YYYYMMDDTHHMMSSZ` timestamp.
- **ISSUE_DATE=<string> as <YYYYMMDD>:** If not specified, defaults to the current date.
- **ISSUE_TIME=<string>:** Issue time as <hhmmssZ> or <hhmmss[+-]HHMM> If not specified, defaults to the current time (in Z timezone).
- **DATASET_DELIVERY_INTERVAL=<string>:** Expected time interval between availability of successive datasets for time-varying data. Must be formatted as `PnYnMnDTnHnMnS` (ISO-8601 duration)
- **TIME_RECORD_INTERVAL=<number>:** Interval in seconds between time records.
- **COMMON_POINT_RULE=[average/low/high/all]:** Defaults to `high`. Procedure used for evaluating the coverage at a position that falls on the boundary or in an area of overlap between geographic objects.
- **UNCERTAINTY_SPEED=<number>:** Uncertainty of speeds in knot
- **UNCERTAINTY_DIRECTION=<number>:** Uncertainty of direction angles in degree
- **HORIZONTAL_POSITION_UNCERTAINTY=<number>:** Horizontal position uncertainty in meter
- **VERTICAL_UNCERTAINTY=<number>:** Vertical uncertainty in meter
- **TIME_UNCERTAINTY=<number>:** Time uncertainty in second
- **COMPRESS=[NONE/DEFLATE]:** Defaults to `DEFLATE`. Compression for elevation and uncertainty grids.
- **ZLEVEL=1-9:** Defaults to `6`. Deflate compression level.
- **BLOCK_SIZE=<integer>:** Chunking size of the HDF5 arrays. Default to 100, or the maximum dimension of the raster if smaller than 100.
- **APPEND_SUBDATASET=[YES/NO]:** Defaults to `NO`. Whether to append the new dataset to an existing S-104 dataset as an extra feature instance group ("SurfaceCurrent.XX")

6.135.4 Validation script

Added in version 3.13.

The Python script validate_s111.py can be used to validate the conformity of a S-111 v2.0 dataset against the specification. It requires the h5py Python module to be installed (typically through "pip install h5py")

Its usage is:

```
$ python validate_s111.py 111TESTXXXX.h5
```

Note that the GDAL S-111 reader is more tolerant that the validation script and can read files with slight non-conformities.

6.135.5 Examples

- Converting a GeoTIFF with surface current speed and direction, with the minimum required metadata items:

```
$ gdal_translate current_speed_and_direction.tif 111TESTXXXX.h5 -of S111 \
    -co TIME_POINT=20251105T012600Z \
    -co DEPTH_TYPE=1 \
    -co VERTICAL_DATUM=MMLW \
    -co VERTICAL_CS=HEIGHT \
    -co SURFACE_CURRENT_DEPTH=0.2 \
    -co DATA_DYNAMICITY=hydrodynamicForecast
```

6.135.6 See Also

- Implemented as s111dataset.cpp.
- S-111 Bathymetric Surface Product Specification
- *BAG driver*
- *S-102 driver*
- *S-104 driver*

6.136 SAFE -- Sentinel-1 SAFE XML Product

Driver short name

SAFE

Driver built-in by default

This driver is built-in by default

Driver for Sentinel products. Currently supports only Sentinel-1 SAR products. See also the *GDAL Sentinel-2 driver*

SENTINEL data products are distributed using a SENTINEL-specific variation of the Standard Archive Format for Europe (SAFE) format specification. The SAFE format has been designed to act as a common format for archiving and conveying data within ESA Earth Observation archiving facilities.

The SAFE driver will be used if the manifest.safe or the containing directory is selected, and it can treat all the imagery as one consistent dataset.

The SAFE driver also reads geolocation grid points from the metadata and represents them as GCPs on the dataset.

ESA will be distributing other satellite datasets in this format; however, at this time this driver only supports specific Sentinel-1 SAR products. All other will be ignored, or result in various runtime errors.

6.136.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.136.2 Multiple measurements

If the product contains multiple measurements (for example multiple polarizations), each one is available as a raster band - if the swath is the same. When the swath is the same, the geographic area is the same.

If the product contains multiple swaths and multiple polarizations, the driver shows the first swath by default. To access other swaths, the user must select a specific subdataset.

The syntax of subdataset naming and their content has been significantly change in GDAL 3.4.

6.136.3 Examples

- Opening the Sentinel-1 product:

```
$ gdalinfo S1A_IW_GRDH_1SDV_20150705T064241_20150705T064306_006672_008EA0_24EE.SAFE/
↪manifest.safe
```

```
Driver: SAFE/Sentinel-1 SAR SAFE Product
Files: S1A_IW_GRDH_1SDV_20150705T064241_20150705T064306_006672_008EA0_24EE.SAFE/
↪manifest.safe
       S1A_IW_GRDH_1SDV_20150705T064241_20150705T064306_006672_008EA0_24EE.SAFE/
↪measurement/s1a-iw-grd-vh-20150705t064241-20150705t064306-006672-008ea0-002.tiff
       S1A_IW_GRDH_1SDV_20150705T064241_20150705T064306_006672_008EA0_24EE.SAFE/
↪measurement/s1a-iw-grd-vv-20150705t064241-20150705t064306-006672-008ea0-001.tiff
Size is 256, 167
Coordinate System is `'
GCP Projection =
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0,
        AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.0174532925199433,
```

(continues on next page)

(continued from previous page)

```
        AUTHORITY["EPSG","9122"]],
    AUTHORITY["EPSG","4326"]]
GCP[  0]: Id=1, Info=
          (0,0) -> (-8.03500070209827,39.6332161725022,141.853266630322)
Metadata:
  ACQUISITION_START_TIME=2015-07-05T06:42:41.504840
  ACQUISITION_STOP_TIME=2015-07-05T06:43:06.503530
  BEAM_MODE=IW
  BEAM_SWATH=IW
  FACILITY_IDENTIFIER=UPA_
  LINE_SPACING=1.000655e+01
  MISSION_ID=S1A
  MODE=IW
  ORBIT_DIRECTION=DESCENDING
  ORBIT_NUMBER=6672
  PIXEL_SPACING=1.000000e+01
  PRODUCT_TYPE=GRD
  SATELLITE_IDENTIFIER=SENTINEL-1
  SENSOR_IDENTIFIER=SAR
  SWATH=IW
Subdatasets:
  SUBDATASET_1_NAME=SENTINEL1_DS:S1A_IW_GRDH_1SDV_20150705T064241_20150705T064306_
↪006672_008EA0_24EE.SAFE:IW_VH
  SUBDATASET_1_DESC=Single band with IW swath and VH polarization
  SUBDATASET_2_NAME=SENTINEL1_DS:S1A_IW_GRDH_1SDV_20150705T064241_20150705T064306_
↪006672_008EA0_24EE.SAFE:IW_VV
  SUBDATASET_2_DESC=Single band with IW swath and VV polarization
  SUBDATASET_3_NAME=SENTINEL1_DS:S1A_IW_GRDH_1SDV_20150705T064241_20150705T064306_
↪006672_008EA0_24EE.SAFE:IW
  SUBDATASET_3_DESC=IW swath with all polarizations as bands
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0,  167.0)
Upper Right (  256.0,    0.0)
Lower Right (  256.0,  167.0)
Center      (  128.0,   83.5)
Band 1 Block=256x16 Type=UInt16, ColorInterp=Undefined
  Metadata:
    POLARISATION=VH
    SWATH=IW
Band 2 Block=256x16 Type=UInt16, ColorInterp=Undefined
  Metadata:
    POLARISATION=VV
    SWATH=IW
```

- It's not mandatory to open manifest.safe, just pass the folder name:

```
$ gdalinfo S1A_IW_GRDH_1SDV_20150705T064241_20150705T064306_006672_008EA0_24EE.SAFE
```

- Opening a single measurement (for example IW/VH):

```
$ gdalinfo SENTINEL1_DS:S1A_IW_GRDH_1SDV_20150705T064241_20150705T064306_006672_
↪008EA0_24EE.SAFE:IW_VV
```

or starting with GDAL 3.4

```
$ gdalinfo SENTINEL1_CALIB:UNCALIB:test.SAFE:IW_VV:AMPLITUDE
```

6.136.4 Data Calibration

Starting with GDAL 3.4, calibration is applied for SIGMA0, BETA0 and GAMMA calibration subdataset

6.136.5 See Also

- SAR Formats (ESA Sentinel Online)
- SAFE Specification (ESA Sentinel Online)
- *GDAL Sentinel-2 driver*

6.137 SAR_CEOS -- CEOS SAR Image

Driver short name

SAR_CEOS

Driver built-in by default

This driver is built-in by default

This is a read-only reader for CEOS SAR image files. To use, select the main imagery file.

This driver works with most Radarsat, JERS-1, JAXA PALSAR-2 ALOS2, JAXA PALSAR-3 ALOS4 and ERS data products, including single look complex products; however, it is unlikely to work for non-Radar CEOS products. The simpler *CEOS -- CEOS Image* driver is often appropriate for these.

This driver will attempt to read 15 lat/long GCPS by sampling the per-scanline CEOS superstructure information. In the case of products from the Alaska Satellite Facility, it will obtain corner coordinates from either the map projection record in the case of ScanSAR products, or the facility data record for non-ScanSAR products. It also captures various pieces of metadata from various header files, including:

```
CEOS_LOGICAL_VOLUME_ID=EERS-1-SAR-MLD
CEOS_FACILITY=CDPF-RSAT
CEOS_PROCESSING_FACILITY=APP
CEOS_PROCESSING_AGENCY=CCRS
CEOS_PROCESSING_COUNTRY=CANADA
CEOS_SOFTWARE_ID=APP 1.62
CEOS_ACQUISITION_TIME=19911029162818919
CEOS_SENSOR_CLOCK_ANGLE=  90.000
CEOS_ELLIPSOID=IUGG_75
CEOS_SEMI_MAJOR=    6378.1400000
CEOS_SEMI_MINOR=    6356.7550000
```

The SAR_CEOS driver also includes some support for SIR-C and PALSAR polarimetric data. The SIR-C format contains an image in compressed scattering matrix form, described here. GDAL decompresses the data as it is read in. The PALSAR format contains bands that correspond almost exactly to elements of the 3x3 Hermitian covariance matrix- see the ERSDAC-VX-CEOS-004A.pdf document for a complete description (pixel storage is described on page

193). GDAL converts these to complex floating point covariance matrix bands as they are read in. The convention used to represent the covariance matrix in terms of the scattering matrix elements HH, HV (=VH), and VV is indicated below. Note that the non-diagonal elements of the matrix are complex values, while the diagonal values are real (though represented as complex bands).

- Band 1: Covariance_11 (Float32) = HH*conj(HH)
- Band 2: Covariance_12 (CFloat32) = sqrt(2)*HH*conj(HV)
- Band 3: Covariance_13 (CFloat32) = HH*conj(VV)
- Band 4: Covariance_22 (Float32) = 2*HV*conj(HV)
- Band 5: Covariance_23 (CFloat32) = sqrt(2)*HV*conj(VV)
- Band 6: Covariance_33 (Float32) = VV*conj(VV)

The identities of the bands are also reflected in the metadata.

NOTE: Implemented as sar_ceosdataset.cpp.

6.137.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.138 SAGA -- SAGA GIS Binary Grid File Format

Driver short name

SAGA

Driver built-in by default

This driver is built-in by default

The driver supports both reading and writing (including create, delete, and copy) SAGA GIS binary grids. SAGA binary grid datasets are made of an ASCII header (.SGRD) and a binary data (.SDAT) file with a common basename. The .SDAT file should be selected to access the dataset. The driver can read compressed .sg-grd-z files that are ZIP archives with .sgrd, .sdat and .prj files.

The driver supports reading the following SAGA datatypes (in brackets the corresponding GDAL types): BIT (GDT_UInt8), BYTE_UNSIGNED (GDT_UInt8), BYTE (GDT_UInt8), SHORTINT_UNSIGNED (GDT_UInt16), SHORTINT (GDT_Int16), INTEGER_UNSIGNED (GDT_UInt32), INTEGER (GDT_Int32), FLOAT (GDT_Float32) and DOUBLE (GDT_Float64).

The driver supports writing the following SAGA datatypes: BYTE_UNSIGNED (GDT_UInt8), SHORTINT_UNSIGNED (GDT_UInt16), SHORTINT (GDT_Int16), INTEGER_UNSIGNED (GDT_UInt32), INTEGER (GDT_Int32), FLOAT (GDT_Float32) and DOUBLE (GDT_Float64).

Currently the driver does not support zFactors other than 1 and reading SAGA grids which are written TOPTOBOTTOM.

NOTE: Implemented as sagadataset.cpp.

6.138.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.139 SENTINEL2 -- Sentinel-2 Products

Driver short name

SENTINEL2

Driver built-in by default

This driver is built-in by default, but requires a JPEG2000 capable driver to read the imagery

Driver for Sentinel-2 Level-1B, Level-1C and Level-2A products. Level-1C with "Safe Compact" encoding are also supported.

The SENTINEL2 driver will be used if the main metadata .xml file at the root of a SENTINEL2 data product is opened (whose name is typically S2A_OPER_MTD_SAFL1C_....xml). It can also accept directly .zip files downloaded from the Copernicus Browser.

To be able to read the imagery, GDAL must be configured with at least one of the JPEG2000 capable drivers.

SENTINEL-2 data are acquired on 13 spectral bands in the visible and near-infrared (VNIR) and Short-wavelength infrared (SWIR) spectrum, as show in the below table:

Band name	Resolution (m)	Central wavelength (nm)	Band width (nm)	Purpose
B01	60	443	20	Aerosol detection
B02	10	490	65	Blue
B03	10	560	35	Green
B04	10	665	30	Red
B05	20	705	15	Vegetation classification
B06	20	740	15	Vegetation classification
B07	20	783	20	Vegetation classification
B08	10	842	115	Near infrared
B08A	20	865	20	Vegetation classification
B09	60	945	20	Water vapour
B10	60	1375	30	Cirrus
B11	20	1610	90	Snow / ice / cloud discrimination
B12	20	2190	180	Snow / ice / cloud discrimination

6.139.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.139.2 Level-1B

Level-1B products are composed of several "granules" of ~ 25 km across-track x ~ 23km along-track, in sensor geometry (i.e. non ortho-rectified). Each granule correspond to the imagery captured by one of the 12 detectors across-track (for a total 290 km swath width). The imagery of each band is put in a separate JPEG2000 file.

Level-1B products are aimed at advanced users.

Starting with GDAL 3.12, two modes are possible, depending on the value of the `L1B_MODE` open option.

6.139.2.1 Default / per-granule mode (`L1B_MODE = GRANULE`)

When opening the main metadata .xml file, the driver will typically expose N * 3 sub-datasets, where N is the number of granules composing the user product, and 3 corresponds to the number of spatial resolutions. There's one for the 4 10m bands, one for the 6 20m bands and one for the 3 60m bands. Caution: the number of such subdatasets can be typically of several hundreds or more.

It is also possible to open the metadata .xml of a given granule, in which case 3 subdatasets will be reported for each of the 3 spatial resolutions.

When opening a subdataset, the georeferencing is made of 5 ground control points for the 4 corner of the images and the center of image.

6.139.2.2 Per-datastrip per-detector per-band mode (L1B_MODE = DATASTRIP)

This mode is the default one since GDAL 3.12 for products that contain geolocation arrays as formatted by the MPC Sen2VM process. Granules participating to the datastrip are virtually mosaiced together, and a geolocation array is associated to them.

Example how to ortho-rectify a subdataset:

```
gdalwarp -overwrite \
     SENTINEL2_L1B_WITH_GEOLOC:"S2B_OPER_MTD_SAFL1B_PDMC_20241022T154709_R023_
↪V20241019T120217_20241019T120235.xml":S2B_OPER_GEO_L1B_DS_2BPS_20241019T153411_
↪S20241019T120215_D09_B05 \
     out_D09_B05.tif -co TILED=YES
```

6.139.3 Level-1C

Level-1C products are organized in ortho-rectified tiles of 100 km x 100 km in UTM WGS84 projections. The imagery of each band is put in a separate JPEG2000 file.

When opening the main metadata .xml file, the driver will typically expose 4 sub-datasets:

- one for the 4 10m bands,
- one for the 6 20m bands,
- one for the 3 60m bands and,
- one for a preview of the R,G,B bands at a 320m resolution

All tiles of same resolution and projection are mosaiced together. If a product spans over several UTM zones, they will be exposed as separate subdatasets.

It is also possible to open the metadata .xml file of each tile (only for original L1C encoding, not supported on "Safe Compact" encoding), in which case the driver will typically expose the 4 above mentioned types of sub-datasets.

6.139.4 Level-2A

Similarly to Level-1C, Level-2A products are organized in ortho-rectified tiles of 100 km x 100 km in UTM WGS84 projections. The imagery of each band is put in a separate JPEG2000 file. The values are Bottom-Of-Atmosphere (BOA) reflectances. L2A specific bands are also computed:

- AOT: Aerosol Optical Thickness map (at 550nm)
- CLD: Raster mask values range from 0 for high confidence clear sky to 100 for high confidence cloudy
- SCL: Scene Classification. The meaning of the values is indicated in the Category Names of the band.
- SNW: Raster mask values range from 0 for high confidence NO snow/ice to 100 for high confidence snow/ice
- WVP: Scene-average Water Vapour map

When opening the main metadata .xml file, the driver will typically expose 4 sub-datasets:

- one for the 4 native 10m bands, and L2A specific bands (AOT and WVP)
- one for the 6 native 20m bands, and L2A specific bands (AOT, WVP, SCL, CLD and SNW),
- one for the B01 and B09 native 60m bands, and L2A specific bands (AOT, WVP, SCL, CLD and SNW),

- one for a true-color image (TCI) of the R,G,B bands at a 10m resolution, for the "compact" L2A product formulation (or a preview of the R,G,B bands at a 320m resolution for the older L2A product formulation). The TCI products at resolution 20m and 60m are not exposed, as they are just subsampled versions of the 10m product.

All tiles of same resolution and projection are mosaiced together. If a product spans over several UTM zones, they will be exposed as separate subdatasets.

6.139.5 Metadata

Metadata of the main metadata .xml file is available in the general metadata domain. The whole XML file is also accessible through the xml:SENTINEL2 metadata domain.

Subdatasets are based on the VRT format, so the definition of this VRT can be obtained by querying the xml:VRT metadata domain.

6.139.6 Performance issues for L1C and L2A

Due to the way Sentinel-2 products are structured, in particular because of the number of JPEG2000 files involved, zoom-out operations can be very slow for products made of many tiles. For interactive display, it can be useful to generate overviews (can be a slow operation by itself). This can be done with the gdaladdo utility on the subdataset name. The overview file is created next to the main metadata .xml file, with the same name, but prefixed with _XX_EPSG_YYYYY.tif.ovr where XX=10m,20m,60m or PREVIEW and YYYYY is the EPSG code.

Trick: if the content of the zoom-out preview is not important for the use case, blank overviews can be created instantaneously by using the NONE resampling method ('-r none' as gdaladdo switch).

When converting a subdataset to another format like tiled GeoTIFF, if using the JP2OpenJPEG driver, the recommended minimum value for the *GDAL_CACHEMAX* configuration option is (subdataset_width * 2048 * 2) / 10000000 if generating a INTERLEAVE=BAND GeoTIFF, or that value multiplied by the number of bands for the default INTERLEAVE=PIXEL configuration. The current versions of the OpenJPEG libraries can also consume a lot of memory to decode a JPEG2000 tile (up to 600MB), so you might want to specify the *GDAL_NUM_THREADS* configuration option to a reasonable number of threads if you are short of memory (the default value is the total number of virtual CPUs).

6.139.7 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The driver can be passed the following open options:

- **ALPHA=[YES/NO]:** Defaults to `NO`. Whether to expose an alpha band. If set, an extra band is added after the Sentinel2 bands with an alpha channel. Its value are:
  - 0 on areas with no tiles, or when the tile data is set to the NODATA or SATURATED special values,
  - 4095 on areas with valid data.
- **L1B_MODE=[DEFAULT/GRANULE/DATASTRIP]:** (GDAL >= 3.12) Defaults to `DEFAULT`. When opening a L1B product, determines what kind of subdatasets it must report. In the DEFAULT mode, if the product contains geolocation arrays as formatted by the MPC Sen2VM process, a subdataset is reported per-datastrip per-detector and per-band will be reported. In the GRANULE mode (only one supported before 3.12), a subdataset is reported per-granule and per-resolution.

6.139.8 Examples

- Opening the main metadata file of a Sentinel2 product:

```
$ gdalinfo S2A_OPER_MTD_SAFL1C_PDMC_20150818T101440_R022_V20150813T102406_
↪20150813T102406.xml
```

```
Driver: SENTINEL2/Sentinel 2
Files: S2A_OPER_MTD_SAFL1C_PDMC_20150818T101440_R022_V20150813T102406_
↪20150813T102406.xml
Size is 512, 512
Coordinate System is `'
Metadata:
  CLOUD_COVERAGE_ASSESSMENT=0.0
  DATATAKE_1_DATATAKE_SENSING_START=2015-08-13T10:10:26.027Z
  DATATAKE_1_DATATAKE_TYPE=INS-NOBS
  DATATAKE_1_ID=GS2A_20150813T101026_000734_N01.03
  DATATAKE_1_SENSING_ORBIT_DIRECTION=DESCENDING
  DATATAKE_1_SENSING_ORBIT_NUMBER=22
  DATATAKE_1_SPACECRAFT_NAME=Sentinel-2A
  DEGRADED_ANC_DATA_PERCENTAGE=0
  DEGRADED_MSI_DATA_PERCENTAGE=0
  FOOTPRINT=POLYGON((11.583573986577191 46.02490454425771, 11.538730738326866 45.
↪03757398414644, 12.93007028286133 44.99812645604949, 12.999359413660665 45.
↪98408391203724, 11.583573986577191 46.02490454425771, 11.583573986577191 46.
↪02490454425771))
  FORMAT_CORRECTNESS_FLAG=PASSED
  GENERAL_QUALITY_FLAG=PASSED
  GENERATION_TIME=2015-08-18T10:14:40.000283Z
  GEOMETRIC_QUALITY_FLAG=PASSED
  PREVIEW_GEO_INFO=BrowseImageFootprint
  PREVIEW_IMAGE_URL=https://pdmcdam2.sentinel2.eo.esa.int/s2pdgs_geoserver/geo_
↪service.php?service=WMS&version=1.1.0&request=GetMap&layers=S2A_A000022_N0103:S2A_
↪A000022_N0103&styles=&bbox=11.538730738326866,44.99812645604949,12.
↪999359413660665,46.02490454425771&width=1579&height=330&srs=EPSG:4326&
↪format=image/png&time=2015-08-13T10:24:06.0Z/2015-08-13T10:24:06.0Z
  PROCESSING_BASELINE=01.03
  PROCESSING_LEVEL=Level-1C
  PRODUCT_START_TIME=2015-08-13T10:24:06.637Z
  PRODUCT_STOP_TIME=2015-08-13T10:24:06.637Z
  PRODUCT_TYPE=S2MSI1C
  QUANTIFICATION_VALUE=1000
  RADIOMETRIC_QUALITY_FLAG=PASSED
  REFERENCE_BAND=B1
  REFLECTANCE_CONVERSION_U=0.973195961910065
  SENSOR_QUALITY_FLAG=PASSED
  SPECIAL_VALUE_NODATA=1
  SPECIAL_VALUE_SATURATED=0
Subdatasets:
  SUBDATASET_1_NAME=SENTINEL2_L1C:S2A_OPER_MTD_SAFL1C_PDMC_20150818T101440_R022_
↪V20150813T102406_20150813T102406.xml:10m:EPSG_32632
  SUBDATASET_1_DESC=Bands B2, B3, B4, B8 with 10m resolution, UTM 32N
  SUBDATASET_2_NAME=SENTINEL2_L1C:S2A_OPER_MTD_SAFL1C_PDMC_20150818T101440_R022_
↪V20150813T102406_20150813T102406.xml:20m:EPSG_32632
  SUBDATASET_2_DESC=Bands B5, B6, B7, B8A, B11, B12 with 20m resolution, UTM 32N
  SUBDATASET_3_NAME=SENTINEL2_L1C:S2A_OPER_MTD_SAFL1C_PDMC_20150818T101440_R022_
↪V20150813T102406_20150813T102406.xml:60m:EPSG_32632
  SUBDATASET_3_DESC=Bands B1, B9, B10 with 60m resolution, UTM 32N
  SUBDATASET_4_NAME=SENTINEL2_L1C:S2A_OPER_MTD_SAFL1C_PDMC_20150818T101440_R022_
↪V20150813T102406_20150813T102406.xml:PREVIEW:EPSG_32632
```

(continues on next page)

(continued from previous page)

```
  SUBDATASET_4_DESC=RGB preview, UTM 32N
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0,  512.0)
Upper Right (  512.0,    0.0)
Lower Right (  512.0,  512.0)
Center      (  256.0,  256.0)
```

- Opening the .zip file directly:

```
$ gdalinfo S2A_OPER_PRD_MSIL1C_PDMC_20150818T101440_R022_V20150813T102406_
↪20150813T102406.zip
```

- Opening the 10 meters resolution bands of a L1C subdataset:

```
$ gdalinfo SENTINEL2_L1C:S2A_OPER_MTD_SAFL1C_PDMC_20150818T101440_R022_
↪V20150813T102406_20150813T102406.xml:10m:EPSG_32632
```

```
Driver: SENTINEL2/Sentinel 2
Files: S2A_OPER_MTD_SAFL1C_PDMC_20150818T101440_R022_V20150813T102406_
↪20150813T102406.xml
       ./GRANULE/S2A_OPER_MSI_L1C_TL_MTI__20150813T201603_A000734_T32TQR_N01.03/S2A_
↪OPER_MTD_L1C_TL_MTI__20150813T201603_A000734_T32TQR.xml
       ./GRANULE/S2A_OPER_MSI_L1C_TL_MTI__20150813T201603_A000734_T32TQR_N01.03/IMG_
↪DATA/S2A_OPER_MSI_L1C_TL_MTI__20150813T201603_A000734_T32TQR_B04.jp2
       ./GRANULE/S2A_OPER_MSI_L1C_TL_MTI__20150813T201603_A000734_T32TQR_N01.03/IMG_
↪DATA/S2A_OPER_MSI_L1C_TL_MTI__20150813T201603_A000734_T32TQR_B03.jp2
       ./GRANULE/S2A_OPER_MSI_L1C_TL_MTI__20150813T201603_A000734_T32TQR_N01.03/IMG_
↪DATA/S2A_OPER_MSI_L1C_TL_MTI__20150813T201603_A000734_T32TQR_B02.jp2
       ./GRANULE/S2A_OPER_MSI_L1C_TL_MTI__20150813T201603_A000734_T32TQR_N01.03/IMG_
↪DATA/S2A_OPER_MSI_L1C_TL_MTI__20150813T201603_A000734_T32TQR_B08.jp2
Size is 10980, 10980
Coordinate System is:
PROJCS["WGS 84 / UTM zone 32N",
    GEOGCS["WGS 84",
        DATUM["WGS_1984",
            SPHEROID["WGS 84",6378137,298.257223563,
                AUTHORITY["EPSG","7030"]],
            AUTHORITY["EPSG","6326"]],
        PRIMEM["Greenwich",0,
            AUTHORITY["EPSG","8901"]],
        UNIT["degree",0.0174532925199433,
            AUTHORITY["EPSG","9122"]],
        AUTHORITY["EPSG","4326"]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",0],
    PARAMETER["central_meridian",9],
    PARAMETER["scale_factor",0.9996],
    PARAMETER["false_easting",500000],
    PARAMETER["false_northing",0],
    UNIT["metre",1,
        AUTHORITY["EPSG","9001"]],
```

(continues on next page)

(continued from previous page)

```
    AXIS["Easting",EAST],
    AXIS["Northing",NORTH],
    AUTHORITY["EPSG","32632"]]
Origin = (699960.000000000000000,5100060.000000000000000)
Pixel Size = (10.000000000000000,-10.000000000000000)
Metadata:
[... same as above ...]
Image Structure Metadata:
  COMPRESSION=JPEG2000
Corner Coordinates:
Upper Left  (  699960.000, 5100060.000) ( 11d35' 0.87"E, 46d 1'29.66"N)
Lower Left  (  699960.000, 4990260.000) ( 11d32'19.43"E, 45d 2'15.27"N)
Upper Right (  809760.000, 5100060.000) ( 12d59'57.69"E, 45d59' 2.70"N)
Lower Right (  809760.000, 4990260.000) ( 12d55'48.25"E, 44d59'53.26"N)
Center      (  754860.000, 5045160.000) ( 12d15'46.56"E, 45d30'48.07"N)
Band 1 Block=128x128 Type=UInt16, ColorInterp=Red
  Description = B4, central wavelength 665 nm
  Overviews: 5490x5490, 2745x2745, 1373x1373, 687x687, 344x344
  Metadata:
    BANDNAME=B4
    BANDWIDTH=30
    BANDWIDTH_UNIT=nm
    SOLAR_IRRADIANCE=1512.79
    SOLAR_IRRADIANCE_UNIT=W/m2/um
    WAVELENGTH=665
    WAVELENGTH_UNIT=nm
  Image Structure Metadata:
    NBITS=12
Band 2 Block=128x128 Type=UInt16, ColorInterp=Green
  Description = B3, central wavelength 560 nm
[...]
Band 3 Block=128x128 Type=UInt16, ColorInterp=Blue
  Description = B2, central wavelength 490 nm
[...]
Band 4 Block=128x128 Type=UInt16, ColorInterp=Undefined
  Description = B8, central wavelength 842 nm
[...]
```

- Conversion of a L1C subdataset to tiled GeoTIFF

```
$ gdal_translate SENTINEL2_L1C:S2A_OPER_MTD_SAFL1C_PDMC_20150818T101440_R022_
↪V20150813T102406_20150813T102406.xml:10m:EPSG_32632 \
                 10m.tif \
                 -co TILED=YES --config GDAL_CACHEMAX 1000 --config GDAL_NUM_
↪THREADS 2
```

- Generating blank overviews for a L1C subdataset:

```
$ gdaladdo -r NONE SENTINEL2_L1C:S2A_OPER_MTD_SAFL1C_PDMC_20150818T101440_R022_
↪V20150813T102406_20150813T102406.xml:10m:EPSG_32632 4
```

- Creating a VRT file from the subdataset (can be convenient to have the subdatasets as files):

```
$ python3 -c "import sys; from osgeo import gdal; ds = gdal.Open(sys.argv[1]);␣
↪open(sys.argv[2], 'wb').write(ds.GetMetadata('xml:VRT')[0].encode('utf-8'))" \
         SENTINEL2_L1C:S2A_OPER_MTD_SAFL1C_PDMC_20150818T101440_R022_
↪V20150813T102406_20150813T102406.xml:10m:EPSG_32632 10m.vrt
```

- Opening the 10 meters resolution bands of a L1B subdataset:

```
$ gdalinfo SENTINEL2_L1B:S2A_OPER_MTD_L1B_GR_SGS__20151024T023555_S20151024T011315_
↪D02.xml:10m
```

```
Driver: SENTINEL2/Sentinel 2
Files: S2A_OPER_MTD_L1B_GR_SGS__20151024T023555_S20151024T011315_D02.xml
       IMG_DATA/S2A_OPER_MSI_L1B_GR_SGS__20151024T023555_S20151024T011315_D02_B04.
↪jp2
       IMG_DATA/S2A_OPER_MSI_L1B_GR_SGS__20151024T023555_S20151024T011315_D02_B03.
↪jp2
       IMG_DATA/S2A_OPER_MSI_L1B_GR_SGS__20151024T023555_S20151024T011315_D02_B02.
↪jp2
       IMG_DATA/S2A_OPER_MSI_L1B_GR_SGS__20151024T023555_S20151024T011315_D02_B08.
↪jp2
Size is 2552, 2304
Coordinate System is `'
GCP Projection =
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0,
        AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.0174532925199433,
        AUTHORITY["EPSG","9122"]],
    AUTHORITY["EPSG","4326"]]
GCP[  0]: Id=, Info=
          (0,0) -> (134.635194391036,-21.4282083310724,0)
GCP[  1]: Id=, Info=
          (0,2304) -> (134.581480136827,-21.6408640426055,0)
GCP[  2]: Id=, Info=
          (2552,2304) -> (134.833308274251,-21.686125031254,0)
GCP[  3]: Id=, Info=
          (2552,0) -> (134.886750925145,-21.4734274382519,0)
GCP[  4]: Id=, Info=
          (1276,1152) -> (134.734115530986,-21.5571457404287,0)
Metadata:
  CLOUDY_PIXEL_PERCENTAGE=0
  DATASTRIP_ID=S2A_OPER_MSI_L1B_DS_SGS__20151024T023555_S20151024T011312_N01.04
  DATATAKE_1_DATATAKE_SENSING_START=2015-10-24T01:13:12.027Z
  DATATAKE_1_DATATAKE_TYPE=INS-NOBS
  DATATAKE_1_ID=GS2A_20151024T011312_001758_N01.04
  DATATAKE_1_SENSING_ORBIT_DIRECTION=DESCENDING
  DATATAKE_1_SENSING_ORBIT_NUMBER=45
  DATATAKE_1_SPACECRAFT_NAME=Sentinel-2A
  DEGRADED_ANC_DATA_PERCENTAGE=0
```

(continues on next page)

(continued from previous page)

```
  DEGRADED_MSI_DATA_PERCENTAGE=0
  DETECTOR_ID=02
  DOWNLINK_PRIORITY=NOMINAL
  FOOTPRINT=POLYGON((134.635194391036 -21.4282083310724, 134.581480136827 -21.
↪6408640426055, 134.833308274251 -21.686125031254, 134.886750925145 -21.
↪4734274382519, 134.635194391036 -21.4282083310724))
  FORMAT_CORRECTNESS_FLAG=PASSED
  GENERAL_QUALITY_FLAG=PASSED
  GENERATION_TIME=2015-11-12T10:55:12.000947Z
  GEOMETRIC_QUALITY_FLAG=PASSED
  GRANULE_ID=S2A_OPER_MSI_L1B_GR_SGS__20151024T023555_S20151024T011315_D02_N01.04
  PREVIEW_GEO_INFO=BrowseImageFootprint
  PREVIEW_IMAGE_URL=https://pdmcdam2.sentinel2.eo.esa.int/s2pdgs_geoserver/geo_
↪service.php?service=WMS&version=1.1.0&request=GetMap&layers=S2A_A000045_N0104:S2A_
↪A000045_N0104&styles=&bbox=133.512786023161,-25.3930035889714,137.184847290108,-
↪21.385906922696&width=1579&height=330&srs=EPSG:4326&format=image/png&time=2015-10-
↪24T01:13:15.0Z/2015-10-24T01:14:13.0Z
  PROCESSING_BASELINE=01.04
  PROCESSING_LEVEL=Level-1B
  PRODUCT_START_TIME=2015-10-24T01:13:15.497656Z
  PRODUCT_STOP_TIME=2015-10-24T01:14:13.70431Z
  PRODUCT_TYPE=S2MSI1B
  RADIOMETRIC_QUALITY_FLAG=PASSED
  SENSING_TIME=2015-10-24T01:13:15.497656Z
  SENSOR_QUALITY_FLAG=PASSED
  SPECIAL_VALUE_NODATA=1
  SPECIAL_VALUE_SATURATED=0
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0, 2304.0)
Upper Right ( 2552.0,    0.0)
Lower Right ( 2552.0, 2304.0)
Center      ( 1276.0, 1152.0)
Band 1 Block=128x128 Type=UInt16, ColorInterp=Red
  Description = B4, central wavelength 665 nm
  Overviews: 1276x1152, 638x576, 319x288, 160x144
  Metadata:
    BANDNAME=B4
    BANDWIDTH=30
    BANDWIDTH_UNIT=nm
    WAVELENGTH=665
    WAVELENGTH_UNIT=nm
  Image Structure Metadata:
    NBITS=12
Band 2 Block=128x128 Type=UInt16, ColorInterp=Green
  Description = B3, central wavelength 560 nm
[...]
Band 3 Block=128x128 Type=UInt16, ColorInterp=Blue
  Description = B2, central wavelength 490 nm
[...]
Band 4 Block=128x128 Type=UInt16, ColorInterp=Undefined
  Description = B8, central wavelength 842 nm
```

(continues on next page)

(continued from previous page)

```
[...]
```

6.139.9 See Also

- Copernicus Data Space Ecosystem
- Sentinel 2 User guide
- Sentinel 2 User Handbook

6.139.10 Credits

Th driver has been developed by Spatialys with initial funding from Centre National d'Etudes Spatiales (CNES)

6.140 SIGDEM -- Scaled Integer Gridded DEM

Driver short name

SIGDEM

Driver built-in by default

This driver is built-in by default

The SIGDEM driver supports reading and writing Scaled Integer Gridded DEM files.

SIGDEM files contain exactly 1 band. The in-memory band data is stored using GDT_Float64.

SIGDEM prefers use of an EPSG ID inside the file for coordinate systems. Only if the spatial reference doesn't have an EPSG ID will a .prj file be written or read.

NOTE: Implemented as sigdemdataset.cpp.

6.140.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.141 SNAP_TIFF -- Sentinel Application Processing GeoTIFF

Added in version 3.10.

Driver short name

SNAP_TIFF

Driver built-in by default

This driver is built-in by default

This driver deals specifically with GeoTIFF files produced by the Sentinel Application Processing (SNAP) toolbox.

Such files are formulated in a way that makes it difficult to read them with the generic *GTiff -- GeoTIFF File Format* driver.

6.141.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.141.2 Georeferencing

SNAP GeoTIFF files contain a geolocation array (stored as a regular grid of GeoTIFF tie points). It is reported in the `GEOLOCATION` metadata domain.

The 4 corners of the geolocation array are also reported as ground control points for faster (but less accurate) georeferencing.

6.141.3 Metadata

Extensive metadata following DIMAP conventions is reported in the `xml:DIMAP` metadata domain.

6.142 SNODAS -- Snow Data Assimilation System

Driver short name

SNODAS

Driver built-in by default

This driver is built-in by default

This is a convenience driver to read Snow Data Assimilation System data. Those files contain Int16 raw binary data. The file to provide to GDAL is the .Hdr file.

Snow Data Assimilation System (SNODAS) Data Products at NSIDC

NOTE: Implemented as snodasdataset.cpp.

6.142.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.143 SRP -- Standard Product Format (ASRP/USRP) (.gen)

Driver short name

SRP

Driver built-in by default

This driver is built-in by default

The ASRP and USRP raster products (as defined by DGIWG) are variations on a common standard product format and are supported for reading by GDAL. ASRP and USRP datasets are made of several files - typically a .GEN, .IMG, .SOU and .QAL file with a common basename. The .IMG file should be selected to access the dataset.

ASRP (in a geographic coordinate system) and USRP (in a UTM/UPS coordinate system) products are single band images with a palette and georeferencing.

the Transmission Header File (.THF) can also be used as an input to GDAL. If the THF references more than one image, GDAL will report the images it is composed of as subdatasets. If the THF references just one image, GDAL will open it directly.

NOTE: Implemented as srpdataset.cpp.

6.143.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.144 SRTMHGT -- SRTM HGT Format

Driver short name

SRTMHGT

Driver built-in by default

This driver is built-in by default

The SRTM HGT driver currently supports the reading of SRTM-3 and SRTM-1 V2 (HGT) files. The files must be named like NXXEYYY.hgt or NXXEYYY[.something].hgt

The driver can directly read .hgt.zip files provided that they are named like NXXEYYY[.something].hgt.zip and contain a NXXEYYY.hgt file. For previous versions, use /vsizip//path/to/NXXEYYY[.something].hgt.zip/NXXEYYY.hgt syntax

The driver does support creating new files, but the input data must be exactly formatted as a SRTM-3 or SRTM-1 cell. That is the size, and bounds must be appropriate for a cell.

Starting with GDAL 3.10, the driver also supports reading and writing 0.5 degree resolution SRTM HGT files.

See Also:

- SRTM documentation
- SRTM FAQ
- SRTM data

NOTE: Implemented as srtmhgtdataset.cpp.

6.144.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.145 STACIT - Spatio-Temporal Asset Catalog Items

Added in version 3.4.

Driver short name

STACIT

Driver built-in by default

This driver is built-in by default

This driver supports opening STAC API ItemCollections, with the input usually being a STAC API search query or the results saved as a JSON file. Items in the response must include projection information following the Projection Extension Specification. It builds a virtual mosaic from the items.

A STACIT dataset which has no subdatasets is actually a *VRT -- GDAL Virtual Format* dataset. Thus, translating it into VRT will result in a VRT file that directly references the items.

Note that STAC API ItemCollections are not the same as STAC Collections. STAC API ItemCollections are GeoJSON FeatureCollections enhanced with STAC entities.

6.145.1 Open syntax

STACIT datasets/subdatasets can be accessed with one of the following syntaxes:

- `filename.json`: local file
- `STACIT:"https://example.com/filename.json"`: remote file or query
- `STACIT:"filename.json":asset=my_asset`: specify the name of the asset GDAL should read (i.e. "visual")
- `STACIT:"filename.json":collection=my_collect,asset=my_asset`: limit to items in a given collection and specify asset to read
- `STACIT:"filename.json":collection=my_collect,asset=my_asset,crs=my_crs`: specify a collection, asset, and limit to items in a given CRS

Starting with GDAL 3.10, specifying the `-if STACIT` option to command line utilities accepting it, or `STACIT` as the only value of the `papszAllowedDrivers` of *GDALOpenEx()*, also forces the driver to recognize the passed filename or URL.

6.145.2 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are supported:

- **MAX_ITEMS=<integer>:** Defaults to `1000`. Maximum number of items fetched. 0=unlimited.
- **COLLECTION=value:** Name of collection to filter items.
- **CRS=value:** Name of CRS to filter items.
- **ASSET=value:** Name of asset to read.
- **RESOLUTION=[AVERAGE/HIGHEST/LOWEST]:** Defaults to `AVERAGE`. Strategy to use to determine dataset resolution.
- **OVERLAP_STRATEGY=[REMOVE_IF_NO_NODATA/USE_ALL/USE_MOST_RECENT]:** (GDAL >= 3.9.1) Defaults to `REMOVE_IF_NO_NODATA`. Strategy to use when the ItemCollections contains overlapping items, and that some items are fully covered by other items that are more recent. Starting with GDAL 3.9.1, the `REMOVE_IF_NO_NODATA` strategy is applied by default. The STACIT virtual mosaic will omit fully covered items, only if no band declares a nodata value. (Note that the determination whether a band has a nodata value of not is done by opening one of the items, and assuming it is representative of the characteristics of the others in the collection).

  This strategy can be forced in all cases by selecting the `USE_MOST_RECENT` strategy (this was the strategy applied prior to 3.9.1)

  The `USE_ALL` strategy always causes all items to be listed in the virtual mosaic, with the most recent ones being rendered on top of the less recent ones.

6.145.3 Subdatasets

If a STACIT JSON file contains several collections, assets or CRS, the driver will return a list of subdataset names to open each of the possible subdatasets.

6.145.4 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.145.5 Examples

List the subdatasets associated to a STAC search on a given collection, bbox and starting from a datetime:

Bash

PowerShell

Python

```
gdalinfo "STACIT:\"https://planetarycomputer.microsoft.com/api/stac/v1/search?
↪collections=naip&bbox=-100,40,-99,41&datetime=2019-01-01T00:00:00Z%2F..\""
# from GDAL 3.11+
gdal raster info "STACIT:\"https://planetarycomputer.microsoft.com/api/stac/v1/search?
↪collections=naip&bbox=-100,40,-99,41&datetime=2019-01-01T00:00:00Z%2F..\""
```

```
gdalinfo 'STACIT:\"https://planetarycomputer.microsoft.com/api/stac/v1/search?
↪collections=naip&bbox=-100,40,-99,41&datetime=2019-01-01T00:00:00Z%2F..\"'
# from GDAL 3.11+
gdal raster info 'STACIT:\"https://planetarycomputer.microsoft.com/api/stac/v1/search?
↪collections=naip&bbox=-100,40,-99,41&datetime=2019-01-01T00:00:00Z%2F..\"'
```

```
from osgeo import gdal

gdal.UseExceptions()

stac_url = "https://planetarycomputer.microsoft.com/api/stac/v1/search"
bbox = "-100,40,-99,41"
datetime = "2019-01-01T00:00:00Z%2F.."
collections = "naip"
url = f"{stac_url}?collections={collections}&bbox={bbox}&datetime={datetime}"

ds = gdal.OpenEx(
    url, allowed_drivers=["STACIT"]
)  # force the STACIT driver or it opens as GeoJSON

print(gdal.Info(ds, format="json"))
```

Open a subdataset returned by the above request:

Bash

PowerShell

Python

```
gdalinfo "STACIT:\"https://planetarycomputer.microsoft.com/api/stac/v1/search?
↪collections=naip&bbox=-100,40,-99,41&datetime=2019-01-01T00:00:00Z%2F..\":asset=image"
# from GDAL 3.11+
gdal raster info "STACIT:\"https://planetarycomputer.microsoft.com/api/stac/v1/search?
↪collections=naip&bbox=-100,40,-99,41&datetime=2019-01-01T00:00:00Z%2F..\":asset=image"
```

```
gdalinfo 'STACIT:\"https://planetarycomputer.microsoft.com/api/stac/v1/search?
↪collections=naip&bbox=-100,40,-99,41&datetime=2019-01-01T00:00:00Z%2F..\":asset=image'
# from GDAL 3.11+
gdal raster info 'STACIT:\"https://planetarycomputer.microsoft.com/api/stac/v1/search?
↪collections=naip&bbox=-100,40,-99,41&datetime=2019-01-01T00:00:00Z%2F..\":asset=image'
```

```

subdatasets = ds.GetSubDatasets()
# get the first subdataset properties
sds_properties = subdatasets[0]
subdataset_url = sds_properties[0]  # a tuple of URL and description
subdataset = gdal.OpenEx(subdataset_url, allowed_drivers=["STACIT"])

print(gdal.Info(subdataset, format="json"))
```

6.145.6 See Also

- *STACTA - Spatio-Temporal Asset Catalog Tiled Assets* documentation page.

6.146 STACTA - Spatio-Temporal Asset Catalog Tiled Assets

Added in version 3.3.

Driver short name

STACTA

Driver built-in by default

This driver is built-in by default

This driver supports opening JSON files following the Spatio-Temporal Asset Catalog Tiled Assets specification. Such JSON file references tiles (also called metatiles), split from a potentially big dataset according to a tiling scheme, with several zoom levels. The driver provides a single raster view, with overviews, of the dataset described by the JSON file. The driver supports metatiles of arbitrary size.

The driver may use the Electro-Optical Extension and, starting with GDAL 3.8.2, the Raster Extension attached to an asset template.

6.146.1 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The following configuration options are available:

- **GDAL_STACTA_SKIP_MISSING_METATILE=value:** See *`SKIP_MISSING_METATILE`* open option.

6.146.2 Open syntax

STACTA datasets/subdatasets can be accessed with one of the following syntaxes:

- `filename.json`: local file
- `STACTA:"https://example.com/filename.json"`: remote file
- `STACTA:"filename.json":my_asset`: specify an asset of a local/remote file
- `STACTA:"filename.json":my_asset:my_tms`: specify an asset and tiling scheme of a local/remote file

The root of the JSON file must be of type `Feature`.

Starting with GDAL 3.10, specifying the `-if STACTA` option to command line utilities accepting it, or `STACTA` as the only value of the `papszAllowedDrivers` of *`GDALOpenEx()`*, also forces the driver to recognize the passed filename or URL.

6.146.3 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are supported:

- **WHOLE_METATILE=[YES/NO.]:** If set to YES, metatiles will be entirely downloaded (into memory). Otherwise by default, if metatiles are bigger than a threshold, they will be accessed in a piece-wise way.
- **SKIP_MISSING_METATILE=[YES/NO]:** If set to YES, metatiles that are missing will be skipped without error, and corresponding area in the dataset will be filled with the nodata value or zero if there is no nodata value. This setting can also be set with the `GDAL_STACTA_SKIP_MISSING_METATILE` configuration option.

6.146.4 Subdatasets

If a STACTA JSON file contains several asset templates and/or tiling scheme, the driver will return a list of subdataset names to open each of the possible subdatasets.

6.146.5 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.146.6 Creation

STACTA JSON files can be created by the *gdal raster tile* program.

6.146.7 See Also

- *STACIT - Spatio-Temporal Asset Catalog Items* documentation page.

6.147 Terragen -- Terragen™ Terrain File

Driver short name

Terragen

Driver built-in by default

This driver is built-in by default

Terragen terrain files store 16-bit elevation values with optional gridspacing (but not positioning). The file extension for Terragen heightfields is "TER" or "TERRAIN" (which in the former case is the same as Leveller, but the driver only recognizes Terragen files). The driver ID is "Terragen". The dataset is file-based and has only one elevation band. Void elevations are not supported. Pixels are considered points.

6.147.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.147.2 Reading

`dataset::GetProjectionRef()` returns a local coordinate system using meters.

`band::GetUnitType()` returns meters.

Elevations are `Int16`. You must use the `band::GetScale()` and `band::GetOffset()` to convert them to meters.

6.147.3 Writing

Use the `Create` call. Set the `MINUSERPIXELVALUE` option (a float) to the lowest elevation of your elevation data, and `MAXUSERPIXELVALUE` to the highest. The units must match the elevation units you will give to `band::SetUnitType()`.

Call `dataset::SetProjection()` and `dataset::SetGeoTransform()` with your coordinate system details. Otherwise, the driver will not encode physical elevations properly. Geographic (degree-based) coordinate systems will be converted to a local meter-based system.

To maintain precision, a best-fit baseheight and scaling will be used to use as much of the 16-bit range as possible.

Elevations are `Float32`.

6.147.4 Roundtripping

Errors per trip tend to be a few centimeters for elevations and up to one or two meters for ground extents if degree-based coordinate systems are written. Large degree-based DEMs incur unavoidable distortions since the driver currently only uses meters.

6.147.5 See Also

- Implemented as terragendataset.cpp.
- See readme.txt for installation and support information.
- Terragen Terrain File Specification.

6.148 TGA -- TARGA Image File Format

Added in version 3.2.

Driver short name

TGA

Driver built-in by default

This driver is built-in by default

The TGA driver currently supports reading TGA 2.0 files.

The driver supports reading 1 (grey-level or paletted), 3 (RGB), and 4 (RGBA, or RGB-undefined) band images. The driver supports both uncompressed or RLE-compressed images. Top-left or bottom-left origins are supported (if the later, the lines are re-order to expose a top-left order). 16-bit RGB encoded images are also supported.

The following metadata items may be reported: IMAGE_ID, AUTHOR_NAME and COMMENTS. The extended area is used to determine if the fourth band is an alpha channel or not.

6.148.1 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.148.2 Links

- Format specification

6.149 TIL -- EarthWatch/DigitalGlobe .TIL

Driver short name

TIL

Driver built-in by default

This driver is built-in by default

NOTE: Implemented as tildataset.cpp.

6.149.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.150 TileDB - TileDB raster

Driver short name

TileDB

Added in version 3.0.

Build dependencies

TileDB (>= 2.7 starting with GDAL 3.7)

GDAL can read and write TileDB arrays through the TileDB library.

The driver relies on the Open Source TileDB library (MIT licensed).

6.150.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.150.2 Open options

The following open options exist:

- **TILEDB_CONFIG=<filename>:** A local file with TileDB configuration options
- **TILEDB_TIMESTAMP=<integer>:** Open array at this timestamp. Should be strictly greater than zero when set.
- **STATS=[YES/NO]:** Defaults to `NO`. Whether TileDB performance statistics should be displayed.
- **TILEDB_ATTRIBUTE=<string>:** Attribute to read from each band

6.150.3 Creation options

The following creation options exist:

- **TILEDB_CONFIG=<filename>:** A local file with TileDB configuration options
- **COMPRESSION=[NONE/GZIP/ZSTD/LZ4/RLE/BZIP2/DOUBLE-DELTA/POSITIVE-DELTA]:** Compression to use. Default is NONE
- **COMPRESSION_LEVEL=<integer>:** Compression level
- **BLOCKXSIZE=<integer>:** Defaults to `256`. Tile width.
- **BLOCKYSIZE=<integer>:** Defaults to `256`. Tile height
- **STATS=[YES/NO]:** Defaults to `NO`. Whether TileDB performance statistics should be displayed.
- **TILEDB_ATTRIBUTE=<string>:** Co-registered file to add as TileDB attributes. Only applicable for interleave types of band or pixel
- **INTERLEAVE=[BAND/PIXEL/ATTRIBUTES]:** Defaults to `BAND`. Indexing order. Influences how multiband rasters are stored.
  - `BAND`: a 3D array is created with the slowest varying dimension being the band.
  - `PIXEL`: a 3D array is created with the fastest varying dimension being the band.
  - `ATTRIBUTES`: a 2D array is created with each band being stored in a separate TileDB attribute.
- **TILEDB_TIMESTAMP=<integer>:** Create array at this timestamp. Should be strictly greater than zero when set.
- **CREATE_GROUP=[YES/NO]:** (GDAL >= 3.10) Defaults to `YES` since 3.10 (NO previously). Whether the dataset should be created within a TileDB group. When the dataset is created within a TileDB group, overviews that may be created are stored as TileDB arrays inside that group, next to the full resolution array. This makes administration of the dataset easier.

  Otherwise, the past default behavior (CREATE_GROUP=NO) is to create the dataset as a TileDB array. Overviews cannot be created in that mode.

6.150.4 Multidimensional API support

Added in version 3.8.

The TileDB driver supports the *Multidimensional Raster Data Model* for reading and writing operations. It requires GDAL to be built and run against TileDB >= 2.15.

The driver supports: - creating and reading groups and subgroups - creating and reading multidimensional dense arrays with a numeric data type - creating and reading numeric or string attributes in groups and arrays - storing an indexing array of a dimension as a TileDB dimension label

The multidimensional API supports reading dense arrays created by the classic raster API of GDAL.

The following multidimensional dataset open options are available:

- **TILEDB_CONFIG=config**: A local file with TileDB configuration options
- **TILEDB_TIMESTAMP=val**: inclusive ending timestamp when opening this array

The following multidimensional dataset creation options are available:

- **TILEDB_CONFIG=config**: A local file with TileDB configuration options
- **TILEDB_TIMESTAMP=val**: Create array at this timestamp. Should be strictly greater than zero when set.

The following array open options are available:

- **TILEDB_TIMESTAMP=val**: inclusive ending timestamp when opening this array

The following array creation options are available:

- **BLOCKSIZE=val1,val2,...,valN**: Block size in pixels
- **COMPRESSION=NONE/GZIP/ZSTD/LZ4/RLE/BZIP2/DOUBLE-DELTA/POSITIVE-DELTA**: Compression to use. Default is NONE
- **COMPRESSION_LEVEL=int_value**: compression level
- **IN_MEMORY=YES/NO**: whether the array should be only in-memory. Useful to create an indexing variable that is serialized as a dimension label
- **TILEDB_TIMESTAMP=val**: Create array at this timestamp. Should be strictly greater than zero when set.

Cf tiledb_multidim.py for examples of how to use the Python multidimensional API with the TileDB driver.

6.150.5 Standalone plugin compilation

Added in version 3.10.

While this driver may be built as part of a whole GDAL build, either in libgdal itself, or as a plugin, it is also possible to only build this driver as a plugin, against an already built libgdal.

The version of the GDAL sources used to build the driver must match the version of the libgdal it is built against.

For example, from a "build_tiledb" directory under the root of the GDAL source tree:

```
cmake -S ../frmts/tiledb -DCMAKE_PREFIX_PATH=/path/to/GDAL_installation_prefix -DTileDB_
↪DIR:PATH=/path/to/tiledb_installation_prefix/lib/cmake/TileDB
cmake --build .
```

Note that such a plugin, when used against a libgdal not aware of it, will be systematically loaded at GDAL driver initialization time, and will not benefit from *deferred plugin loading capabilities <rfc-96>*. For that, libgdal itself must be built with the CMake variable GDAL_REGISTER_DRIVER_TILEDB_FOR_LATER_PLUGIN=ON set.

6.150.6 See Also

- TileDB home page
- *TileDB vector* documentation page

6.151 TSX -- TerraSAR-X Product

Driver short name

TSX

Driver built-in by default

This driver is built-in by default

NOTE: Implemented as tsxdataset.cpp.

6.151.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.152 USGSDEM -- USGS ASCII DEM (and CDED)

Driver short name

USGSDEM

Driver built-in by default

This driver is built-in by default

GDAL includes support for reading USGS ASCII DEM files. This is the traditional format used by USGS before being replaced by SDTS, and is the format used for CDED DEM data products from Canada. Most popular variations on USGS DEM files should be supported, including correct recognition of coordinate system, and georeferenced positioning.

The 7.5 minute (UTM grid) USGS DEM files will generally have regions of missing data around the edges, and these are properly marked with a nodata value. Elevation values in USGS DEM files may be in meters or feet, and this will be indicated by the return value of GDALRasterBand::GetUnitType() (either "m" or "ft").

Note that USGS DEM files are represented as one big tile. This may cause cache thrashing problems if the GDAL tile cache size is small. It will also result in a substantial delay when the first pixel is read as the whole file will be ingested.

Some of the code for implementing usgsdemdataset.cpp was derived from VTP code by Ben Discoe. See the Virtual Terrain project for more information on VTP.

6.152.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

NOTE: Implemented as usgsdemdataset.cpp.

The USGS DEM reading code in GDAL was derived from the importer in the VTP software. The export capability was developed with the financial support of the Yukon Department of Environment.

6.153 VICAR -- VICAR

Driver short name

VICAR

Driver built-in by default

This driver is built-in by default

Note

PDS3 datasets can incorporate a VICAR header. By default, GDAL will use the *PDS* driver in that situation. Starting with GDAL 3.1, if the *GDAL_TRY_PDS3_WITH_VICAR* configuration option is set to YES, the dataset will be opened by the VICAR driver. Alternatively, starting with GDAL 3.10, specifying the `-if VICAR` option to command line utilities accepting it, or `VICAR` as the only value of the `papszAllowedDrivers` of *GDALOpenEx()*, also forces the driver to recognize the passed filename.

6.153.1 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.153.2 Metadata

Starting with GDAL 3.1, the VICAR label can be retrieved as JSON-serialized content in the json:VICAR metadata domain.

For example:

```
$ python3
from osgeo import gdal
ds = gdal.Open('../autotest/gdrivers/data/test_vicar_truncated.bin')
print(ds.GetMetadata_List('json:VICAR')[0])
{
 "LBLSIZE":9680,
 "FORMAT":"BYTE",
 "TYPE":"IMAGE",
 "BUFSIZ":2097152,
 "DIM":3,
 "EOL":0,
 "RECSIZE":4840,
 "ORG":"BSQ",
 "NL":1000,
 "NS":400,
 "NB":1,
 "N1":4000,
 "N2":1000,
 "N3":1,
 "N4":0,
 "NBB":0,
 "NLB":0,
 "HOST":"X86-64-LINX",
 "INTFMT":"LOW",
 "REALFMT":"RIEEE",
 "BHOST":"X86-LINUX",
 "BINTFMT":"LOW",
 "BREALFMT":"RIEEE",
 "BLTYPE":"M94_HRSC",
 "COMPRESS":"NONE",
 "EOCI1":0,
 "EOCI2":0,
 "PROPERTY":{
     "M94_ORBIT":{
         "ORBIT_NUMBER":5273,
         "ASCENDING_NODE_LONGITUDE":118.46,
         "ORBITAL_ECCENTRICITY":1.23,
```

(continues on next page)

(continued from previous page)

```
        "ORBITAL_INCLINATION":4.56,
        "PERIAPSIS_ARGUMENT_ANGLE":7.89,
        "PERIAPSIS_TIME":"PERIAPSIS_TIME",
        "PERIAPSIS_ALTITUDE":333.16,
        "ORBITAL_SEMIMAJOR_AXIS":1.23,
        "SPACECRAFT_SOLAR_DISTANCE":4.56,
        "SPACECRAFT_CLOCK_START_COUNT":"1\/1",
        "SPACECRAFT_CLOCK_STOP_COUNT":"1\/2",
        "START_TIME":"start_time",
        "STOP_TIME":"stop_time",
        "SPACECRAFT_POINTING_MODE":"NADIR",
        "RIGHT_ASCENSION":-1.00000000000000001e+32,
        "DECLINATION":-1.0000000000000001e+32,
        "OFFSET_ANGLE":-1.00000000000000001e+32,
        "SPACECRAFT_ORIENTATION":[
            0.000000,
            -1.000000,
            0.000000
        ]
    },
    [...]
    "PHOT":{
        "PHO_FUNC":"NONE"
    }
},
"TASK":{
    "HRCONVER":{
        "USER":"mexsyst",
        "DAT_TIM":"DAT_TIM",
        "SPICE_FILE_NAME":[
            "foo"
        ],
        "SPICE_FILE_ID":"(LSK,SCLK,ON)",
        "DETECTOR_TEMPERATURE":1.23,
        "DETECTOR_TEMPERATURE__UNIT":"degC",
        "FOCAL_PLANE_TEMPERATURE":8.5833,
        "FOCAL_PLANE_TEMPERATURE__UNIT":"degC",
        "INSTRUMENT_TEMPERATURE":2.34,
        "INSTRUMENT_TEMPERATURE__UNIT":"degC",
        "LENS_TEMPERATURE":4.56,
        "LENS_TEMPERATURE__UNIT":"degC",
        "SOURCE_FILE_NAME":"SOURCE_FILE_NAME",
        "MISSING_FRAMES":0,
        "OVERFLOW_FRAMES":0,
        "ERROR_FRAMES":1
    }
  }
}
```

or

```
$ gdalinfo -json ../autotest/gdrivers/data/test_vicar_truncated.bin -mdd all
```

6.153.3 Binary prefixes

Starting with GDAL 3.1, if the VICAR label declares a non-zero binary prefix length (*NBB* label item), then GDAL will look in the *vicar.json* configuration file if there is an entry corresponding to the *BLTYPE* label item (currently only M94_HRSC is defined), and if there is a match, a OGR vector layer will be available on the dataset, with a feature for each image record.

For example:

```
$ ogrinfo h0038_0000.bl2.16 -al -q

Layer name: binary_prefixes
OGRFeature(binary_prefixes):0
    EphTime (Real) = 127988268.646895
    Exposure (Real) = 40.1072692871094
    COT (Integer) = 28275
    FEETemp (Integer) = 28508
    FPMTemp (Integer) = 29192
    OBTemp (Integer) = 28295
    FERT (Integer) = 27001
    LERT (Integer) = 28435
    CmpDataLen (Integer) = 146
    FrameCount (Integer) = 486
    Pischel (Integer) = 5
    ActPixel (Integer) = 5120
    RSHits (Integer) = 0
    DceInput (Integer) = 0
    DceOutput (Integer) = 4
    FrameErr1 (Integer) = 0
    FrameErr2 (Integer) = 0
    Gob1 (Integer) = 0
    Gob2 (Integer) = 0
    Gob3 (Integer) = 0
    DSS (Integer) = 97
    DecmpErr1 (Integer) = 0
    DecmpErr2 (Integer) = 0
    DecmpErr3 (Integer) = 0
    FillerFlag (Integer) = 5
```

6.153.4 Creation support

Starting with GDAL 3.1, the VICAR driver supports updating imagery of existing datasets, creating new datasets through the CreateCopy() and Create() interfaces.

When using CreateCopy(), gdal_translate or gdalwarp, an effort is made to preserve as much as possible of the original label when doing VICAR to VICAR conversions. This can be disabled with the *USE_SRC_LABEL=NO* creation option.

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The available creation options are:

- **GEOREF_FORMAT=[MIPL/GEOTIFF]:** (GDAL >= 3.4) How to encode georeferencing information. Defaults to MIPL using the `MAP` property group. When setting to GEOTIFF, a `GEOTIFF` property group will be used using GeoTIFF keys and tags. The COORDINATE_SYSTEM_NAME, POSITIVE_LONGITUDE_DIRECTION and TARGET_NAME options will be ignored when selecting the GEOTIFF encoding.
- **COORDINATE_SYSTEM_NAME=[PLANETOCENTRIC/PLANETOGRAPHIC]:** Defaults to

PLANETOCENTRIC. Value of MAP.COORDINATE_SYSTEM_NAME. If specified, and USE_SRC_MAP is in effect, this will be taken into account to override the source COORDINATE_SYSTEM_NAME.

- **POSITIVE_LONGITUDE_DIRECTION=[EAST/WEST]:** Defaults to EAST. Value of MAP.override. If specified, and USE_SRC_MAP is in effect, this will be taken into account to override the source POSITIVE_LONGITUDE_DIRECTION.

TARGET_NAME=value:

- Value of MAP.TARGET_NAME. This is normally deduced from the SRS datum name. If specified, and USE_SRC_MAP is in effect, this will be taken into account to override the source TARGET_NAME.
- **USE_SRC_LABEL=[YES/NO.]:** Defaults to YES. Whether to use source label in VICAR to VICAR conversions.
- **LABEL=value:** Label to use, either as a JSON string or a filename containing one. If defined, takes precedence over *USE_SRC_LABEL*.
- **COMPRESS=[NONE/BASIC/BASIC2.]:** Defaults to NONE. Compression method. For maximum interoperability, do not use BASIC or BASIC2 which are not well specified and not always available in VICAR capable applications.

6.153.5 See Also

- Implemented as vicardataset.cpp.
- VICAR documentation
- VICAR file format

6.154 VRT -- GDAL Virtual Format

Driver short name

VRT

Driver built-in by default

This driver is built-in by default

6.154.1 Introduction

GDAL Virtual Format is a raster driver that allows a virtual GDAL dataset to be composed from other GDAL datasets with repositioning, and algorithms potentially applied as well as various kinds of metadata altered or added. VRT descriptions of datasets can be saved in an XML format normally given the extension .vrt. The *gdal raster mosaic* or *gdal raster stack* programs can build a VRT from a list of datasets.

Note .vrt files starting with

- <OGRVRTDataSource> open with *ogrinfo*, etc.
- <VRTDataset> open with *gdalinfo*, etc.

The VRT format can also describe *Warped VRT* and *Pansharpened VRT*

For mosaic with a very large number of tiles (tens of thousands or mores), the *GTI* driver may be used starting with GDAL 3.9.

An example of a simple .vrt file referring to a 512x512 dataset with one band loaded from `utm.tif` might look like this:

```
<VRTDataset rasterXSize="512" rasterYSize="512">
    <GeoTransform>440720.0, 60.0, 0.0, 3751320.0, 0.0, -60.0</GeoTransform>
    <VRTRasterBand dataType="Byte" band="1">
        <ColorInterp>Gray</ColorInterp>
        <SimpleSource>
        <SourceFilename relativeToVRT="1">utm.tif</SourceFilename>
        <SourceBand>1</SourceBand>
        <SrcRect xOff="0" yOff="0" xSize="512" ySize="512"/>
        <DstRect xOff="0" yOff="0" xSize="512" ySize="512"/>
        </SimpleSource>
    </VRTRasterBand>
</VRTDataset>
```

Many aspects of the VRT file are a direct XML encoding of the *Raster Data Model* which should be reviewed for understanding of the semantics of various elements.

VRT files can be produced by translating to VRT format. The resulting file can then be edited to modify mappings, add metadata or other purposes. VRT files can also be produced programmatically by various means.

This tutorial will cover the .vrt file format (suitable for users editing .vrt files), and how .vrt files may be created and manipulated programmatically for developers.

6.154.2 Creation options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following creation options are supported:

- **BLOCKXSIZE=value:** (GDAL >= 3.7) Sets block width.
- **BLOCKYSIZE=value:** (GDAL >= 3.7) Sets block height. Setting explicitly the block size is an advanced setting that should only be used when the user has determined that it is needed. By default the block size is set to:
  - 128x128 for a source-based VRT raster band. Unless the VRT is made of a single source and this single source is not subsetted, in which case the block size of the unique source will be set as the VRT raster band block size)
  - 512x128 for a warped VRT.

6.154.3 .vrt Format

A XML schema of the GDAL VRT format is available.

Note, .vrt files starting with - <OGRVRTDataSource> open with ogrinfo, etc. - <VRTDataset> open with gdalinfo, etc.

Virtual files stored on disk are kept in an XML format with the following elements.

VRTDataset: This is the root element for the whole GDAL dataset. It must have the attributes rasterXSize and rasterYSize describing the width and height of the dataset in pixels. It may have a subClass attributes with values VRTWarpedDataset (*Warped VRT*) or VRTPansharpenedDataset (*Pansharpened VRT*). It may have SRS, GeoTransform, GCPList, Metadata, MaskBand and VRTRasterBand subelements.

```
<VRTDataset rasterXSize="512" rasterYSize="512">
```

6.154.3.1 VRTDataset

The allowed subelements for VRTDataset are :

- **SRS**: This element contains the spatial reference system (coordinate system) in OGC WKT format. Note that this must be appropriately escaped for XML, so items like quotes will have the ampersand escape sequences substituted. As well as WKT, valid input to the *OGRSpatialReference::SetFromUserInput()* method (such as well known GEOGCS names, and PROJ.4 format) is also allowed in the SRS element.

```
<SRS dataAxisToSRSAxisMapping="1,2">PROJCS[&quot;NAD27 / UTM zone 11N&quot;,GEOGCS[&quot;
↪NAD27&quot;,DATUM[&quot;North_American_Datum_1927&quot;,SPHEROID[&quot;Clarke 1866&
↪quot;,6378206.4,294.9786982139006,AUTHORITY[&quot;EPSG&quot;,&quot;7008&quot;]],
↪AUTHORITY[&quot;EPSG&quot;,&quot;6267&quot;]],PRIMEM[&quot;Greenwich&quot;,0],UNIT[&
↪quot;degree&quot;,0.0174532925199433],AUTHORITY[&quot;EPSG&quot;,&quot;4267&quot;]],
↪PROJECTION[&quot;Transverse_Mercator&quot;],PARAMETER[&quot;latitude_of_origin&quot;,
↪0],PARAMETER[&quot;central_meridian&quot;,-117],PARAMETER[&quot;scale_factor&quot;,0.
↪9996],PARAMETER[&quot;false_easting&quot;,500000],PARAMETER[&quot;false_northing&quot;,
↪0],UNIT[&quot;metre&quot;,1,AUTHORITY[&quot;EPSG&quot;,&quot;9001&quot;]],AUTHORITY[&
↪quot;EPSG&quot;,&quot;26711&quot;]]</SRS>
```

The **dataAxisToSRSAxisMapping** attribute is allowed since GDAL 3.0 to describe the relationship between the axis indicated in the CRS definition and the axis of the GeoTransform or GCP metadata. The value of the attribute is a comma separated list of integers. The number of elements of this list must be the number of axis of the CRS. Values start at 1. If m denotes the array values of this attribute, then m[0] is the data axis number for the first axis of the CRS. If the attribute is missing, then the OAMS_TRADITIONAL_GIS_ORDER data axis to CRS axis mapping strategy is implied.

- **GeoTransform**: This element contains a six value affine geotransformation for the dataset, mapping between pixel/line coordinates and georeferenced coordinates. Typically (geotransform[0], geotransform[3]) will be the (easting, northing) of the upper-left corner of the raster, geotransform[1] the horizontal resolution in geospatial coordinates/pixel, and geotransform[5] the vertical resolution in geospatial coordinates/pixel, as a negative value if the image is north-up oriented. See *Affine GeoTransform* for more details about that mapping.

```
<GeoTransform>440720.0,  60,  0.0,  3751320.0,  0.0, -60.0</GeoTransform>
```

- **GCPList**: This element contains a list of Ground Control Points for the dataset, mapping between pixel/line coordinates and georeferenced coordinates. The Projection attribute should contain the SRS of the georeferenced coordinates in the same format as the SRS element. The dataAxisToSRSAxisMapping attribute is the same as in the SRS element.

```
<GCPList Projection="EPSG:4326">
    <GCP Id="1" Info="a" Pixel="0.5" Line="0.5" X="0.0" Y="0.0" Z="0.0" />
    <GCP Id="2" Info="b" Pixel="13.5" Line="23.5" X="1.0" Y="2.0" Z="0.0" />
</GCPList>
```

- **Metadata**: This element contains a list of metadata name/value pairs associated with the VRTDataset as a whole, or a VRTRasterBand. It has <MDI> (metadata item) subelements which have a "key" attribute and the value as the data of the element. The Metadata element can be repeated multiple times, in which case it must be accompanied with a "domain" attribute to indicate the name of the metadata domain.

```
<Metadata>
  <MDI key="md_key">Metadata value</MDI>
</Metadata>
```

- **MaskBand**: This element represents a mask band that is shared between all bands on the dataset (see GMF_PER_DATASET in RFC 15). It must contain a single VRTRasterBand child element, that is the de-

scription of the mask band itself.

```
<MaskBand>
  <VRTRasterBand dataType="Byte">
    <SimpleSource>
      <SourceFilename relativeToVRT="1">utm.tif</SourceFilename>
      <SourceBand>mask,1</SourceBand>
      <SrcRect xOff="0" yOff="0" xSize="512" ySize="512"/>
      <DstRect xOff="0" yOff="0" xSize="512" ySize="512"/>
    </SimpleSource>
  </VRTRasterBand>
</MaskBand>
```

- **OverviewList**: (GDAL >= 3.2.0, not valid for VRTPansharpenedDataset) This elements contains a list of overview factors, separated by space, to create "virtual overviews". For example `2 4`. It can be used so that bands of the VRT datasets declare overviews. This only makes sense to use if the sources added in those bands have themselves overviews compatible with the declared factor. It is generally not necessary to use this mechanism, since downsampling pixel requests on a VRT dataset/band are able to use overviews of the sources, even when the VRT bands do not declare them. One situation where explicit overviews are needed at the VRT level is the warping of a VRT to a lower resolution. This element can also be used with an existing VRT dataset by running *`GDALDataset::BuildOverviews()`* or **gdaladdo** with the *`VRT_VIRTUAL_OVERVIEWS`* configuration option set to `YES`:

  VRT_VIRTUAL_OVERVIEWS=[YES/NO]: Defaults to `NO`.

  Virtual overviews have the least priority compared to the **Overview** element at the **VRTRasterBand** level, or to materialized .vrt.ovr files.

- **VRTRasterBand**: This represents one band of a dataset.

6.154.3.2 VRTRasterBand

The attributes for VRTRasterBand are:

- **dataType** (optional): type of the pixel data associated with this band (use names Byte, Int8, UInt16, Int16, UInt32, Int32, UInt64, Int64, Float32, Float64, CInt16, CInt32, CFloat32 or CFloat64). If not specified, defaults to 1
- **band** (optional): band number this element represents (1 based).
- **blockXSize** (optional, GDAL >= 3.3): block width. If not specified, defaults to the minimum of the raster width and 128.
- **blockYSize** (optional, GDAL >= 3.3): block height. If not specified, defaults to the minimum of the raster height and 128.

This element may have Metadata, ColorInterp, NoDataValue, HideNoDataValue, ColorTable, GDALRasterAttributeTable, Description and MaskBand subelements as well as the various kinds of source elements such as SimpleSource, ComplexSource, AveragedSource, NoDataFromMaskSource, KernelFilteredSource and ArraySource. A raster band may have many "sources" indicating where the actual raster data should be fetched from, and how it should be mapped into the raster bands pixel space.

The allowed subelements for VRTRasterBand are :

- **ColorInterp**: The data of this element should be the name of a color interpretation type. One of Gray, Palette, Red, Green, Blue, Alpha, Hue, Saturation, Lightness, Cyan, Magenta, Yellow, Black, or Unknown.

```
<ColorInterp>Gray</ColorInterp>:
```

- **NoDataValue**: Specifies the NoData value that the raster band will report. If not specified, NoData pixels from the band's sources (as well as pixels not covered by any source) will be set to zero. If the source rasters have their own NoData values that differ from the one specified in `<NoDataValue>`, a `ComplexSource` with a `<NODATA>` element can be used to convert source NoData pixels to the value specified here.

```
<NoDataValue>-100.0</NoDataValue>
```

- **HideNoDataValue**: If this value is `YES/TRUE/1`, the NoData value specified by `<NoDataValue>` will not be reported. Essentially, the reader will not be aware of a NoData pixel when it reads one. Any datasets copied/translated from this will not have a NoData value. This is useful when you want to specify a fixed background value for the dataset. (The background will be the value specified by the NoDataValue element.) Default value is `NO` when this element is absent.

```
<HideNoDataValue>1</HideNoDataValue>
```

- **ColorTable**: This element is parent to a set of Entry elements defining the entries in a color table. Currently only RGBA color tables are supported with c1 being red, c2 being green, c3 being blue and c4 being alpha. The entries are ordered and will be assumed to start from color table entry 0.

```
<ColorTable>
  <Entry c1="0" c2="0" c3="0" c4="255"/>
  <Entry c1="145" c2="78" c3="224" c4="255"/>
</ColorTable>
```

- **GDALRasterAttributeTable**: (GDAL >=2.3) This element is parent to a set of FieldDefn elements defining the columns of a raster attribute table, followed by a set of Row elements defining the values of the columns of each row.

```
<GDALRasterAttributeTable>
  <FieldDefn index="0">
    <Name>Value</Name>
    <Type>0</Type>
    <Usage>0</Usage>
  </FieldDefn>
  <FieldDefn index="1">
    <Name>Red</Name>
    <Type>0</Type>
    <Usage>6</Usage>
  </FieldDefn>
  <FieldDefn index="2">
    <Name>Green</Name>
    <Type>0</Type>
    <Usage>7</Usage>
  </FieldDefn>
  <FieldDefn index="3">
    <Name>Blue</Name>
    <Type>0</Type>
    <Usage>8</Usage>
  </FieldDefn>
  <Row index="0">
    <F>-500</F>
    <F>127</F>
    <F>40</F>
    <F>65</F>
```

(continues on next page)

(continued from previous page)

```
  </Row>
  <Row index="1">
    <F>-400</F>
    <F>154</F>
    <F>168</F>
    <F>118</F>
  </Row>
</GDALRasterAttributeTable>
```

- **Description**: This element contains the optional description of a raster band as its text value.

```
<Description>Crop Classification Layer</Description>
```

- **UnitType**: This optional element contains the vertical units for elevation band data. One of "m" for meters or "ft" for feet. Default assumption is meters.

```
<UnitType>ft</UnitType>
```

- **Offset**: This optional element contains the offset that should be applied when computing "real" pixel values from scaled pixel values on a raster band. The default is 0.0.

```
<Offset>0.0</Offset>
```

- **Scale**: This optional element contains the scale that should be applied when computing "real" pixel values from scaled pixel values on a raster band. The default is 1.0.

```
<Scale>1.0</Scale>
```

- **Overview**: This optional element describes one overview level for the band. It should have a child SourceFilename and SourceBand element. The SourceFilename may have a relativeToVRT boolean attribute. Multiple elements may be used to describe multiple overviews.

```
<Overview>
  <SourceFilename relativeToVRT="1">yellowstone_2.1.ntf.r2</SourceFilename>
  <SourceBand>1</SourceBand>
</Overview>
```

- **CategoryNames**: This optional element contains a list of Category subelements with the names of the categories for classified raster band.

```
<CategoryNames>
  <Category>Missing</Category>
  <Category>Non-Crop</Category>
  <Category>Wheat</Category>
  <Category>Corn</Category>
  <Category>Soybeans</Category>
</CategoryNames>
```

- **SimpleSource**: The *SimpleSource* indicates that raster data should be read from a separate dataset, indicating the dataset, and band to be read from, and how the data should map into this band's raster space.
- **AveragedSource**: The AveragedSource is derived from the SimpleSource and shares the same properties except that it uses an averaging resampling instead of a nearest neighbour algorithm as in SimpleSource, when the size of the destination rectangle is not the same as the size of the source rectangle. Note: a more general mechanism to specify resampling algorithms can be used. See above paragraph about the 'resampling' attribute.

- **NoDataFromMaskSource**: (GDAL >= 3.9) The NoDataFromMaskSource is derived from the SimpleSource and shares the same properties except that it replaces the value of the source with the value of the NODATA child element when the value of the mask band of the source is less or equal to the MaskValueThreshold child element. This is typically used to transform a R,G,B,A image into a R,G,B one with a NoData value.
- **ComplexSource**: The *ComplexSource* is derived from the SimpleSource (so it shares the SourceFilename, SourceBand, SrcRect and DstRect elements), but it provides support to rescale and offset the range of the source values. Certain regions of the source can be masked by specifying the NODATA value, or starting with GDAL 3.3, with the <UseMaskBand>true</UseMaskBand> element.
- **KernelFilteredSource**: The *KernelFilteredSource* is a pixel source derived from the Simple Source (so it shares the SourceFilename, SourceBand, SrcRect and DstRect elements, but it also passes the data through a simple filtering kernel specified with the Kernel element.
- **ArraySource**: The *ArraySource* indicates that raster data should be read from a 2D array using the multidimensional API.
- **MaskBand**: This element represents a mask band that is specific to the VRTRasterBand it contains. It must contain a single VRTRasterBand child element, that is the description of the mask band itself.

Sources

SimpleSource

The SimpleSource may have the SourceFilename, SourceBand, SrcRect, and DstRect subelements. The SrcRect element will indicate what rectangle on the indicated source file should be read, and the DstRect element indicates how that rectangle of source data should be mapped into the VRTRasterBands space.

SrcRect and DstRect are expressed in pixel/line coordinate space. Their relationship with the geospatial coordinate space is given by the geotransform matrix of the source for SrcRect, and of the VRT itself (**GeoTransform** element) for DstRect. See *Affine GeoTransform* for more details about that mapping.

The relativeToVRT attribute on the SourceFilename indicates whether the filename should be interpreted as relative to the .vrt file (value is 1) or not relative to the .vrt file (value is 0). The default is 0.

Some characteristics of the source band can be specified in the optional `SourceProperties` element to enable the VRT driver to defer the opening of the source dataset until it really needs to read data from it. This is particularly useful when building VRTs with a big number of source datasets. The needed parameters are the raster dimensions, the size of the blocks and the data type. If the SourceProperties tag is not present, the source dataset will be opened at the same time as the VRT itself.

> **Note**
>
> Starting with GDAL 3.4, the `SourceProperties` element is no longer necessary for deferred opening of the source datasets.

The content of the SourceBand subelement can refer to a mask band. For example mask,1 means the mask band of the first band of the source.

```
<SimpleSource>
  <SourceFilename relativeToVRT="1">utm.tif</SourceFilename>
  <SourceBand>1</SourceBand>
  <SourceProperties RasterXSize="512" RasterYSize="512" DataType="Byte" BlockXSize="128"␣
↪BlockYSize="128"/>
  <SrcRect xOff="0" yOff="0" xSize="512" ySize="512"/>
  <DstRect xOff="0" yOff="0" xSize="512" ySize="512"/>
</SimpleSource>
```

A OpenOptions subelement can be added to specify the open options to apply when opening the source dataset. It has <OOI> (open option item) subelements which have a "key" attribute and the value as the data of the element.

```
<SimpleSource>
  <SourceFilename relativeToVRT="1">utm.tif</SourceFilename>
  <OpenOptions>
      <OOI key="OVERVIEW_LEVEL">0</OOI>
  </OpenOptions>
  <SourceBand>1</SourceBand>
  <SourceProperties RasterXSize="256" RasterYSize="256" DataType="Byte" BlockXSize="128"␣
↪BlockYSize="128"/>
  <SrcRect xOff="0" yOff="0" xSize="256" ySize="256"/>
  <DstRect xOff="0" yOff="0" xSize="256" ySize="256"/>
</SimpleSource>
```

A resampling attribute can be specified on a SimpleSource or ComplexSource element to specified the resampling algorithm used when the size of the destination rectangle is not the same as the size of the source rectangle. The values allowed for that attribute are : nearest,bilinear,cubic, cubicspline,lanczos,average,mode.

```
<SimpleSource resampling="cubic">
  <SourceFilename relativeToVRT="1">utm.tif</SourceFilename>
  <SourceBand>1</SourceBand>
  <SourceProperties RasterXSize="256" RasterYSize="256" DataType="Byte" BlockXSize="128"␣
↪BlockYSize="128"/>
  <SrcRect xOff="0" yOff="0" xSize="256" ySize="256"/>
  <DstRect xOff="0" yOff="0" xSize="128" ySize="128"/>
</SimpleSource>
```

Starting with GDAL 3.11, it is also possible to use a in-line VRTDataset as the source by using the VRTDataset element instead of SourceFilename.

```
<SimpleSource>
  <VRTDataset rasterXSize="20" rasterYSize="20">
    <VRTRasterBand dataType="Byte" band="1">
      <SimpleSource>
        <SourceFilename relativeToVRT="1">../byte.tif</SourceFilename>
        <SourceBand>1</SourceBand>
        <SourceProperties RasterXSize="20" RasterYSize="20" DataType="Byte" BlockXSize=
↪"20" BlockYSize="20" />
        <SrcRect xOff="0" yOff="0" xSize="20" ySize="20" />
        <DstRect xOff="0" yOff="0" xSize="20" ySize="20" />
      </SimpleSource>
    </VRTRasterBand>
  </VRTDataset>
  <SourceBand>1</SourceBand>
  <SourceProperties RasterXSize="20" RasterYSize="20" DataType="Byte" BlockXSize="20"␣
↪BlockYSize="20" />
  <SrcRect xOff="0" yOff="0" xSize="20" ySize="20" />
  <DstRect xOff="0" yOff="0" xSize="20" ySize="20" />
</SimpleSource>
```

ComplexSource

Alternatively to linear scaling, non-linear scaling using a power function can be used by specifying the Exponent, SrcMin, SrcMax, DstMin and DstMax elements. If SrcMin and SrcMax are not specified, they are computed from the source minimum and maximum value (which might require analyzing the whole source dataset). Exponent must be positive. (Those 5 values can be set with the -exponent and -scale options of gdal_translate.)

The ComplexSource supports adding a custom lookup table to transform the source values to the destination. The LUT can be specified using the following form:

```
<LUT>[src value 1]:[dest value 1],[src value 2]:[dest value 2],...</LUT>
```

The intermediary values are calculated using a linear interpolation between the bounding destination values of the corresponding range. Source values should be listed in a monotonically non-decreasing order. If there is a Not-A-Number (NaN) source value, it should be the first one. Clamping is performed for input pixel values outside of the range specified by the LUT. That is, if an input pixel value is lower than the minimum source value, then the destination value corresponding to that minimum source value is used as the output pixel value. And similarly for an input pixel value that is greater than the maximum source value. To remap values without interpolation, the "reclassify" pixel function can be used. See *Derived Bands (pixel functions)*.

The ComplexSource supports fetching a color component from a source raster band that has a color table. The ColorTableComponent value is the index of the color component to extract : 1 for the red band, 2 for the green band, 3 for the blue band or 4 for the alpha band.

When transforming the source values the operations are executed in the following order:

- Masking, if the NODATA element is set or, starting with GDAL 3.3, if the UseMaskBand is set to true and the source band has a mask band. Note that this is binary masking only, so no alpha blending is done if the mask band is actually an alpha band with non-0 or non-255 values.
- Color table expansion
- For linear scaling, applying the scale ratio, then scale offset
- For non-linear scaling, apply (DstMax-DstMin) * pow((SrcValue-SrcMin) / (SrcMax-SrcMin), Exponent) + DstMin
- Table lookup

```
<ComplexSource>
  <SourceFilename relativeToVRT="1">utm.tif</SourceFilename>
  <SourceBand>1</SourceBand>
  <ScaleOffset>0</ScaleOffset>
  <ScaleRatio>1</ScaleRatio>
  <ColorTableComponent>1</ColorTableComponent>
  <LUT>0:0,2345.12:64,56789.5:128,2364753.02:255</LUT>
  <NODATA>0</NODATA>  <!-- if the mask is a mask or alpha band, use <UseMaskBand>true</
↪UseMaskBand> -->
  <SrcRect xOff="0" yOff="0" xSize="512" ySize="512"/>
  <DstRect xOff="0" yOff="0" xSize="512" ySize="512"/>
</ComplexSource>
```

Non-linear scaling:

```
<ComplexSource>
  <SourceFilename relativeToVRT="1">16bit.tif</SourceFilename>
  <SourceBand>1</SourceBand>
  <Exponent>0.75</Exponent>
```

(continues on next page)

(continued from previous page)

```
  <SrcMin>0</SrcMin>
  <SrcMax>65535</SrcMax>
  <DstMin>0</DstMin>
  <DstMax>255</DstMax>
  <SrcRect xOff="0" yOff="0" xSize="512" ySize="512"/>
  <DstRect xOff="0" yOff="0" xSize="512" ySize="512"/>
</ComplexSource>
```

KernelFilteredSource

The Kernel element should have two child elements, Size and Coefs and optionally the boolean attribute normalized (defaults to false=0). The size must always be an odd number, and the Coefs must have Size * Size entries separated by spaces. For now kernel is not applied to sub-sampled or over-sampled data.

At the top edge, the values of the first row are replicated to virtually extend the source window by a number of rows equal to the radius of the kernel. And similarly for the bottom, left and right edges. This strategy may potentially lead to unexpected results depending on the applied kernel.

```
<KernelFilteredSource>
  <SourceFilename>/debian/home/warmerda/openev/utm.tif</SourceFilename>
  <SourceBand>1</SourceBand>
  <Kernel normalized="1">
    <Size>3</Size>
    <Coefs>0.11111111 0.11111111 0.11111111 0.11111111 0.11111111 0.11111111 0.11111111␣
↪0.11111111 0.11111111</Coefs>
  </Kernel>
</KernelFilteredSource>
```

A separable kernel may also be used. In this case the number of Coefs entries should correspond to the Size. The Coefs specify a one-dimensional kernel which is applied along each axis in succession, resulting in far quicker execution. Many common image-processing filters are separable. For example, a Gaussian blur:

```
<KernelFilteredSource>
  <SourceFilename>/debian/home/warmerda/openev/utm.tif</SourceFilename>
  <SourceBand>1</SourceBand>
  <Kernel normalized="1">
    <Size>13</Size>
    <Coefs>0.01111 0.04394 0.13534 0.32465 0.60653 0.8825 1.0 0.8825 0.60653 0.32465 0.
↪13534 0.04394 0.01111</Coefs>
  </Kernel>
</KernelFilteredSource>
```

Starting with GDAL 3.12, a Function element can be set as a child of KernelFilteredSource and take values `min`, `max`, `stddev`, `median` or `mode`.

For example to compute the median value in a 3x3 neighborhood around each pixel:

```
<KernelFilteredSource>
  <SourceFilename>/debian/home/warmerda/openev/utm.tif</SourceFilename>
  <SourceBand>1</SourceBand>
  <Kernel>
    <Size>3</Size>
    <Coefs>1 1 1
```

(continues on next page)

(continued from previous page)

```
            1 1 1
            1 1 1</Coefs>
  </Kernel>
  <Function>median</Function>
</KernelFilteredSource>
```

NoDataFromMaskSource

Added in version 3.9.

The NoDataFromMaskSource is derived from the SimpleSource and shares the same properties except that it replaces the value of the source with the value of the NODATA child element when the value of the mask band of the source is less or equal to the MaskValueThreshold child element. This is typically used to transform a R,G,B,A image into a R,G,B one with a NoData value. An optional RemappedValue element can be set to specify the value onto which valid pixels whose value is the one of NODATA should be remapped to. When RemappedValue is not explicitly specified, for Byte bands, if NODATA=255, it is implicitly set to 254, otherwise it is set to NODATA+1.

```
<NoDataFromMaskSource>
  <SourceFilename relativeToVRT="1">in.tif</SourceFilename>
  <SourceBand>1</SourceBand>
  <MaskValueThreshold>128</MaskValueThreshold> <!-- if the mask value is &lt;= 128,␣
↪pixels are set to NODATA=0 -->
  <NODATA>0</NODATA>
  <RemappedValue>1</RemappedValue> <!-- valid/unmasked pixels at NODATA=0 are remapped␣
↪to 1 -->
</NoDataFromMaskSource>
```

ArraySource

Added in version 3.8.

The *ArraySource* indicates that raster data should be read from a 2D array using the multidimensional API. If the original array is not a 2D array, a DerivedArray with a View step must be typically used to create a 2D slice.

ArraySource can have the following child elements:

- `SingleSourceArray`, `DerivedArray` or `Array`: required as first child element
- `SrcRect` (same syntax and semantics as in SimpleSource): optional
- `DstRect` (same syntax and semantics as in SimpleSource): optional

SingleSourceArray

SingleSourceArray references a multidimensional raster through a SourceFilename element and an array through its full path within it with SourceArray. When SingleSourceArray is used as a direct child of ArraySource, it must be a 2-dimensional array.

Example:

```
<ArraySource>
  <SingleSourceArray>
    <SourceFilename relativeToVRT="1">byte.nc</SourceFilename>
    <SourceArray>/my_array</SourceArray>
```

(continues on next page)

(continued from previous page)

```
  </SingleSourceArray>
</ArraySource>
```

Be careful that quite often arrays in the multidimensional API use a "bottom-up" approach, and it might thus be needed to reverse the Y axis, with a View expression with a DerivedArray (cf below example)

DerivedArray

A DerivedArray starts with an array coming from a SingleSourceArray (but it could also be a DerivedArray or a Array), which must be the first child element, and applies different processing steps wrapped in zero or more Step elements.

- View step: applies the *GDALMDArray::GetView()* method

The `View` element must have an `expr` attribute, which is the argument of *GDALMDArray::GetView()* method

The following example slices the 3D dimension at index 0 and flips the vertical axis (assuming 1st dimension is a non-spatial one, 2nd dimension is the Y dimension and 3rd dimension is the X dimension)

```
<ArraySource>
  <DerivedArray>
      <SingleSourceArray>
        <SourceFilename relativeToVRT="1">3d_array.nc</SourceFilename>
        <SourceArray>/my_array</SourceArray>
      </SingleSourceArray>
      <Step>
          <View expr="[0,::-1,...]"/>
      </Step>
   </DerivedArray>
</ArraySource>
```

- `Transpose` step: applies the *GDALMDArray::Transpose()* method

The `Transpose` element must have a `newOrder` attribute, which is the argument of *GDALMDArray::Transpose()* method

The following example transposes the 2 axis of a 2D array

```
<ArraySource>
  <DerivedArray>
      <SingleSourceArray>
        <SourceFilename relativeToVRT="1">2d_array.nc</SourceFilename>
        <SourceArray>/my_array</SourceArray>
      </SingleSourceArray>
      <Step>
          <Transpose newOrder="[1,0]"/>
      </Step>
   </DerivedArray>
</ArraySource>
```

- Resample step: applies the *GDALMDArray::GetResampled()* method

The `Resample` element may have the following child elements, which correspond to the arguments of the *GDALMDArray::GetResampled()* method:

- `Dimension`: with a `name` and `size` attribute. This element may be repeated.

- `ResampleAlg`: whose value is `NearestNeighbour` (default), `Cubic`, `CubicSpline`, `Bilinear`, `Lanczos`, `Average`, `RMS` or `Mode`
- `SRS`: target SRS (string recognized by *`OGRSpatialReference::SetFromUserInput()`*)
- `Option` with a `name` attribute and the value as the content of the element. This element may be repeated.

```
<ArraySource>
  <DerivedArray>
      <SingleSourceArray>
        <SourceFilename relativeToVRT="1">test.nc</SourceFilename>
        <SourceArray>/Band1</SourceArray>
      </SingleSourceArray>
      <Step>
          <Resample>
            <Dimension name="Y" size="40"/>
            <Dimension name="X" size="40"/>
            <ResampleAlg>Bilinear</ResampleAlg>
            <SRS>EPSG:4267</SRS>
            <Option name="IGNORED">YES</Option>
          </Resample>
      </Step>
   </DerivedArray>
</ArraySource>
```

- Grid step: applies the *`GDALMDArray::GetGridded()`* method

The `Grid` element may have the following child elements, which correspond to the arguments of the *`GDALMDArray::GetGridded()`* method:

- `GridOptions` (required): name and options of the gridding algorithm
- `XArray` (optional): its child value should be a SingleSourceArray, DerivedArray or Array pointing to the array to use for the X dimension. This is needed if the source array does not have a `coordinates` attribute.
- `YArray` (optional): its child value should be a SingleSourceArray, DerivedArray or Array pointing to the array to use for the Y dimension. This is needed if the source array does not have a `coordinates` attribute.
- `Option` with a `name` attribute and the value as the content of the element. This element may be repeated.

```
<ArraySource>
  <DerivedArray>
      <SingleSourceArray>
        <SourceFilename relativeToVRT="1">test.nc</SourceFilename>
        <SourceArray>/Band1</SourceArray>
      </SingleSourceArray>
      <Step>
          <Grid>
              <GridOptions>invdist</GridOptions>
              <XArray>
                  <SingleSourceArray>
                    <SourceFilename relativeToVRT="1">test.nc</SourceFilename>
                    <SourceArray>/varX</SourceArray>
                  </SingleSourceArray>
              </XArray>
              <YArray>
                  <SingleSourceArray>
```

(continues on next page)

(continued from previous page)

```
                    <SourceFilename relativeToVRT="1">test.nc</SourceFilename>
                    <SourceArray>/varY</SourceArray>
                  </SingleSourceArray>
              </YArray>
              <Option name="IGNORED">YES</Option>
          </Grid>
      </Step>
   </DerivedArray>
</ArraySource>
```

- `GetUnscaled` step: applies the *`GDALMDArray::GetUnscaled()`* method

It does not have any child elements.

```
<ArraySource>
  <DerivedArray>
      <SingleSourceArray>
        <SourceFilename relativeToVRT="1">2d_array.nc</SourceFilename>
        <SourceArray>/my_array</SourceArray>
      </SingleSourceArray>
      <Step>
          <GetUnscaled/>
      </Step>
   </DerivedArray>
</ArraySource>
```

- `GetMask` step: applies the *`GDALMDArray::GetMask()`* method

The `GetMask` element may have the following child elements, which correspond to the arguments of the *`GDALMDArray::GetMask()`* method:

- `Option` with a `name` attribute and the value as the content of the element. This element may be repeated.

```
<ArraySource>
  <DerivedArray>
      <SingleSourceArray>
        <SourceFilename relativeToVRT="1">2d_array.nc</SourceFilename>
        <SourceArray>/my_array</SourceArray>
      </SingleSourceArray>
      <Step>
          <GetMask>
              <Option name="UNMASK_FLAGS">microwave,land</Option>
          </GetMask>
      </Step>
   </DerivedArray>
</ArraySource>
```

Array

Array can be used to mosaic several multidimensional arrays.

6.154.4 Overviews

GDAL can make efficient use of overviews available in the sources that compose the bands when dealing with RasterIO() requests that involve downsampling. But in the general case, the VRT bands themselves will not expose overviews.

Except if (from top priority to lesser priority) :

- The **Overview** element is present in the VRTRasterBand element. See above.
- or external .vrt.ovr overviews are built
- (starting with GDAL 3.2) explicit virtual overviews, if a **OverviewList** element is declared in the VRTDataset element (see above). Those virtual overviews will be hidden by external .vrt.ovr overviews that might be built later.
- Implicit virtual overviews, if the VRTRasterBand are made of a single SimpleSource or ComplexSource that has overviews. Those virtual overviews will be hidden by external .vrt.ovr overviews that might be built later.

6.154.5 .vrt Descriptions for Raw Files

> **Warning**
>
> Consult the *Restricting access to Raw Files* below section for potential security issues related to that functionality and how to restrict it.

So far we have described how to derive new virtual datasets from existing files supported by GDAL. However, it is also common to need to utilize raw binary raster files for which the regular layout of the data is known but for which no format specific driver exists. This can be accomplished by writing a .vrt file describing the raw file.

For example, the following .vrt describes a raw raster file containing floating point complex pixels in a file called *l2p3hhsso.img*. The image data starts from the first byte (ImageOffset=0). The byte offset between pixels is 8 (PixelOffset=8), the size of a CFloat32. The byte offset from the start of one line to the start of the next is 9376 bytes (LineOffset=9376) which is the width (1172) times the size of a pixel (8).

```
<VRTDataset rasterXSize="1172" rasterYSize="1864">
    <VRTRasterBand dataType="CFloat32" band="1" subClass="VRTRawRasterBand">
        <SourceFilename relativetoVRT="1">l2p3hhsso.img</SourceFilename>
        <ImageOffset>0</ImageOffset>
        <PixelOffset>8</PixelOffset>
        <LineOffset>9376</LineOffset>
        <ByteOrder>MSB</ByteOrder>
    </VRTRasterBand>
</VRTDataset>
```

Some things to note are that the VRTRasterBand has a subClass specifier of "VRTRawRasterBand". Also, the VRTRawRasterBand contains a number of previously unseen elements but no "source" information. VRTRawRasterBands may never have sources (i.e. SimpleSource), but should contain the following elements in addition to all the normal "metadata" elements previously described which are still supported.

- **SourceFilename**: The name of the raw file containing the data for this band. The relativeToVRT attribute can be used to indicate if the SourceFilename is relative to the .vrt file (1) or not (0).
- **ImageOffset**: The offset in bytes to the beginning of the first pixel of data of this image band. Defaults to zero.
- **PixelOffset**: The offset in bytes from the beginning of one pixel and the next on the same line. In packed single band data this will be the size of the **dataType** in bytes.

- **LineOffset**: The offset in bytes from the beginning of one scanline of data and the next scanline of data. In packed single band data this will be PixelOffset * rasterXSize.
- **ByteOrder**: Defines the byte order of the data on disk. Either LSB (Least Significant Byte first) such as the natural byte order on Intel x86 systems or MSB (Most Significant Byte first) such as the natural byte order on Motorola or Sparc systems. Defaults to being the local machine order.

A few other notes:

- The image data on disk is assumed to be of the same data type as the band **dataType** of the VRTRawRasterBand.
- All the non-source attributes of the VRTRasterBand are supported, including color tables, metadata, nodata values, and color interpretation.
- The VRTRawRasterBand supports in place update of the raster, whereas the source based VRTRasterBand is always read-only.
- Multiple bands in the one .vrt file can come from the same raw file. Just ensure that the ImageOffset, PixelOffset, and LineOffset definition for each band is appropriate for the pixels of that particular band.

Another example, in this case a 400x300 RGB pixel interleaved image.

```
<VRTDataset rasterXSize="400" rasterYSize="300">
<VRTRasterBand dataType="Byte" band="1" subClass="VRTRawRasterBand">
    <ColorInterp>Red</ColorInterp>
    <SourceFilename relativetoVRT="1">rgb.raw</SourceFilename>
    <ImageOffset>0</ImageOffset>
    <PixelOffset>3</PixelOffset>
    <LineOffset>1200</LineOffset>
</VRTRasterBand>
<VRTRasterBand dataType="Byte" band="2" subClass="VRTRawRasterBand">
    <ColorInterp>Green</ColorInterp>
    <SourceFilename relativetoVRT="1">rgb.raw</SourceFilename>
    <ImageOffset>1</ImageOffset>
    <PixelOffset>3</PixelOffset>
    <LineOffset>1200</LineOffset>
</VRTRasterBand>
<VRTRasterBand dataType="Byte" band="3" subClass="VRTRawRasterBand">
    <ColorInterp>Blue</ColorInterp>
    <SourceFilename relativetoVRT="1">rgb.raw</SourceFilename>
    <ImageOffset>2</ImageOffset>
    <PixelOffset>3</PixelOffset>
    <LineOffset>1200</LineOffset>
</VRTRasterBand>
</VRTDataset>
```

6.154.6 Restricting access to Raw Files

Some usages of GDAL, for example its use on a server that allows users to upload a (VRT) file, convert it to another format, and get the result back, could be abused to read the content of local files. Starting with GDAL 3.12, it is possible to restrict the use of the VRTRawRasterBand capability in several ways:

- at build time, the CMake `GDAL_VRT_ENABLE_RAWRASTERBAND` variable can be set to `OFF`, to complete disable VRTRawRasterBand.
- at runtime, with the following configuration options:
  - **GDAL_VRT_ENABLE_RAWRASTERBAND=[YES/NO]:** (GDAL >= 3.12) Defaults to `YES`. Whether the VRTRawRasterBand capability is allowed at runtime.

- **GDAL_VRT_RAWRASTERBAND_ALLOWED_SOURCE=[SIBLING_OR_CHILD_OF_VRT_PATH /ONLY_REMOTE/ALL/<path>]:** (GDAL >= 3.12) Defaults to `SIBLING_OR_CHILD_OF_VRT_PATH`. Restricts which SourceFilename values are allowed:
  * if set to `SIBLING_OR_CHILD_OF_VRT_PATH` (which is the default value of that configuration option starting with GDAL 3.12), the `relativeToVRT` attribute of `SourceFilename` will need to set to `1`, and the path expressed by `SourceFilename` must not contain any ../ or ..\ substring. Note that GDAL does not try to detect if one of the files, in the file hierarchy below the directory of the VRT, is a symbolic link pointing to elsewhere in the file system.
  * if set to `ONLY_REMOTE`, only `SourceFilename` pointing to one of the *VSI network based file systems* will be allowed. Be careful though that this could still be used to access files accessible on a local network, depending on the network configuration of the machine on which GDAL is run.
  * if set to `ALL` there is no restriction on the value of `SourceFilename`.
  * if set to one absolute path (or several ones, separated by the `;` (semi-colon) character on Windows, or `:` (colon) on other operating systems), only `SourceFilename` that start with those allowed absolute paths will be accepted.

In versions before GDAL 3.12, disabling entirely the VRT driver by setting the *GDAL_SKIP* configuration option to `VRT` may be a workaround.

6.154.7 Creation of VRT Datasets

The VRT driver supports several methods of creating VRT datasets. The `vrtdataset.h` include file should be installed with the core GDAL include files, allowing direct access to the VRT classes. However, even without that most capabilities remain available through standard GDAL interfaces.

To create a VRT dataset that is a clone of an existing dataset use the *GDALDriver::CreateCopy()* method. For example to clone `utm.tif` into a `wrk.vrt` file in C++ the following could be used:

```
GDALDriver *poDriver = (GDALDriver *) GDALGetDriverByName( "VRT" );
GDALDataset *poSrcDS, *poVRTDS;

poSrcDS = (GDALDataset *) GDALOpenShared( "utm.tif", GA_ReadOnly );

poVRTDS = poDriver->CreateCopy( "wrk.vrt", poSrcDS, FALSE, NULL, NULL, NULL );

GDALClose((GDALDatasetH) poVRTDS);
GDALClose((GDALDatasetH) poSrcDS);
```

Note the use of *GDALOpenShared()* when opening the source dataset. It is advised to use *GDALOpenShared()* in this situation so that you are able to release the explicit reference to it before closing the VRT dataset itself. In other words, in the previous example, you could also invert the 2 last lines, whereas if you open the source dataset with *GDALOpen()*, you'd need to close the VRT dataset before closing the source dataset.

To obtain the resulting VRT XML of `wrk.vrt` without having to read the text from an actual file, you can modify the above code to open the new dataset with an empty filename and use the "xml:VRT" metadata domain.

```
// no filename
poVRTDS = poDriver->CreateCopy( "", poSrcDS, FALSE, NULL, NULL, NULL );

// obtain the actual XML text that a VRT file would contain
const char *xmlvrt = poVRTDS->GetMetadata("xml:VRT")[0];
```

To create a virtual copy of a dataset with some attributes added or changed such as metadata or coordinate system that are often hard to change on other formats, you might do the following. In this case, the virtual dataset is created

"in memory" only by virtual of creating it with an empty filename, and then used as a modified source to pass to a *GDALDriver::CreateCopy()* written out in TIFF format.

```
poVRTDS = poDriver->CreateCopy( "", poSrcDS, FALSE, NULL, NULL, NULL );

poVRTDS->SetMetadataItem( "SourceAgency", "United States Geological Survey");
poVRTDS->SetMetadataItem( "SourceDate", "July 21, 2003" );

poVRTDS->GetRasterBand( 1 )->SetNoDataValue( -999.0 );

GDALDriver *poTIFFDriver = (GDALDriver *) GDALGetDriverByName( "GTiff" );
GDALDataset *poTiffDS;

poTiffDS = poTIFFDriver->CreateCopy( "wrk.tif", poVRTDS, FALSE, NULL, NULL, NULL );

GDALClose((GDALDatasetH) poTiffDS);
```

In the above example the nodata value is set as -999. You can set the HideNoDataValue element in the VRT dataset's band using *GDALRasterBand::SetMetadataItem()* on that band.

```
poVRTDS->GetRasterBand( 1 )->SetMetadataItem( "HideNoDataValue" , "1" );
```

In this example a virtual dataset is created with the *GDALDriver::Create()* method, and adding bands and sources programmatically, but still via the "generic" API. A special attribute of VRT datasets is that sources can be added to the VRTRasterBand (but not to VRTRawRasterBand) by passing the XML describing the source into *GDALRasterBand::SetMetadataItem()* on the special domain target "new_vrt_sources". The domain target "vrt_sources" may also be used, in which case any existing sources will be discarded before adding the new ones. In this example we construct a simple averaging filter source instead of using the simple source.

```
// construct XML for simple 3x3 average filter kernel source.
const char *pszFilterSourceXML  =
"<KernelFilteredSource>"
"  <SourceFilename>utm.tif</SourceFilename><SourceBand>1</SourceBand>"
"  <Kernel>"
"    <Size>3</Size>"
"    <Coefs>0.111 0.111 0.111 0.111 0.111 0.111 0.111 0.111 0.111</Coefs>"
"  </Kernel>"
"</KernelFilteredSource>";

// Create the virtual dataset.
poVRTDS = poDriver->Create( "", 512, 512, 1, GDT_UInt8, NULL );
poVRTDS->GetRasterBand(1)->SetMetadataItem("source_0",pszFilterSourceXML,
                                           "new_vrt_sources");
```

A more general form of this that will produce a 3x3 average filtered clone of any input datasource might look like the following. In this case we deliberately set the filtered datasource as in the "vrt_sources" domain to override the SimpleSource created by the *cpp:func:`GDALDriver::CreateCopy* method. The fact that we used *cpp:func:GDALDriver::CreateCopy* ensures that all the other metadata, georeferencing and so forth is preserved from the source dataset ... the only thing we are changing is the data source for each band.

```
int   nBand;
GDALDriver *poDriver = (GDALDriver *) GDALGetDriverByName( "VRT" );
GDALDataset *poSrcDS, *poVRTDS;
```

(continues on next page)

(continued from previous page)

```
poSrcDS = (GDALDataset *) GDALOpenShared( pszSourceFilename, GA_ReadOnly );

poVRTDS = poDriver->CreateCopy( "", poSrcDS, FALSE, NULL, NULL, NULL );

for( nBand = 1; nBand <= poVRTDS->GetRasterCount(); nBand++ )
{
    char szFilterSourceXML[10000];

    GDALRasterBand *poBand = poVRTDS->GetRasterBand( nBand );

    sprintf( szFilterSourceXML,
        "<KernelFilteredSource>"
        "  <SourceFilename>%s</SourceFilename><SourceBand>%d</SourceBand>"
        "  <Kernel>"
        "    <Size>3</Size>"
        "    <Coefs>0.111 0.111 0.111 0.111 0.111 0.111 0.111 0.111 0.111</Coefs>"
        "  </Kernel>"
        "</KernelFilteredSource>",
        pszSourceFilename, nBand );

    poBand->SetMetadataItem( "source_0", szFilterSourceXML, "vrt_sources" );
}
```

The `VRTDataset` class is one of the few dataset implementations that supports the *GDALDataset::AddBand()* method. The options passed to the *GDALDataset::AddBand()* method can be used to control the type of the band created (VRTRasterBand, VRTRawRasterBand, VRTDerivedRasterBand), and in the case of the VRTRawRasterBand to set its various parameters. For standard VRTRasterBand, sources should be specified with the above *GDALRasterBand::SetMetadataItem()* examples.

```
GDALDriver *poDriver = (GDALDriver *) GDALGetDriverByName( "VRT" );
GDALDataset *poVRTDS;

poVRTDS = poDriver->Create( "out.vrt", 512, 512, 0, GDT_UInt8, NULL );
char** papszOptions = NULL;
papszOptions = CSLAddNameValue(papszOptions, "subclass", "VRTRawRasterBand"); // if not␣
↪specified, default to VRTRasterBand
papszOptions = CSLAddNameValue(papszOptions, "SourceFilename", "src.tif"); // mandatory
papszOptions = CSLAddNameValue(papszOptions, "ImageOffset", "156"); // optional. default␣
↪= 0
papszOptions = CSLAddNameValue(papszOptions, "PixelOffset", "2"); // optional. default =␣
↪size of band type
papszOptions = CSLAddNameValue(papszOptions, "LineOffset", "1024"); // optional. default␣
↪= size of band type * width
papszOptions = CSLAddNameValue(papszOptions, "ByteOrder", "LSB"); // optional. default =␣
↪machine order
papszOptions = CSLAddNameValue(papszOptions, "relativeToVRT", "true"); // optional.␣
↪default = false
poVRTDS->AddBand(GDT_UInt8, papszOptions);
CSLDestroy(papszOptions);

delete poVRTDS;
```

6.154.8 Derived Bands (pixel functions)

A "derived" band is a specialized type of band that derives its pixel values from its source bands using a "pixel function" at the time that values are read from the raster. Using derived bands you can create VRT datasets that manipulate bands on the fly without having to create new band files on disk. GDAL provides a number of *built-in pixel functions*; additional pixel functions can be defined in *C++* or *Python* and registered with GDAL using a unique key.

A VRTRasterBand can be made a VRTDerivedRasterBand by setting attribute subClass="VRTDerivedRasterBand".

Some of the common subelements for VRTRasterBand (whose subClass="VRTDerivedRasterBand") are listed here. They can be used with built-in, C++, or Python pixel functions.

- **PixelFunctionType**: (required): A pixel function with this name must be defined.
- **SkipNonContributingSources**: (optional, added in GDAL 3.7, defaults to false) = true/false: Whether sources that do not intersect the VRTRasterBand RasterIO() requested region should be omitted. By default, data for all sources, including ones that do not intersect it, are passed to the pixel function. By setting this parameter to true, only sources that intersect the requested region will be passed.

6.154.8.1 Example 1: Calculating a derived band

For example, you might want to generate a band using four source bands from a nine-band input dataset (x0, x3, x4, and x8) and some constant y:

```
band_value = sqrt((x3*x3+x4*x4)/(x0*x8)) + y;
```

This can be accomplished using the built-in "expression" pixel function. The following VRT XML can be used:

```
<VRTDataset rasterXSize="1000" rasterYSize="1000">
    <VRTRasterBand dataType="Float32" band="1" subClass="VRTDerivedRasterBand">
        <Description>Magnitude</Description>
        <PixelFunctionType>expression</PixelFunctionType>
        <PixelFunctionArguments expression="sqrt((x3*x3+x4*x4)/(x0*x8)) + 4" />
        <SimpleSource name="x0">
            <SourceFilename relativeToVRT="1">nine_band.dat</SourceFilename>
            <SourceBand>1</SourceBand>
        </SimpleSource>
        <SimpleSource name="x3">
            <SourceFilename relativeToVRT="1">nine_band.dat</SourceFilename>
            <SourceBand>4</SourceBand>
        </SimpleSource>
        <SimpleSource name="x4">
            <SourceFilename relativeToVRT="1">nine_band.dat</SourceFilename>
            <SourceBand>5</SourceBand>
        </SimpleSource>
        <SimpleSource name="x8">
            <SourceFilename relativeToVRT="1">nine_band.dat</SourceFilename>
            <SourceBand>9</SourceBand>
        </SimpleSource>
    </VRTRasterBand>
</VRTDataset>
```

See also

VRT processed datasets, which provide a means of calculating multiple bands at once.

6.154.8.2 Built-in Pixel Functions

GDAL provides a set of default pixel functions that can be used without writing new code:

PixelFunctionType	Number of input sources	PixelFunctionArguments	Description
abs	1		(GDAL >= 3.14) Returns the absolute value
area	0		(GDAL >= 3.13) Returns the area of each pixel in square meters.
argmax	>= 1	`propagat` (optional, default=false	(GDAL >= 3.12) Index (1-based, contrary to `numpy.argmax`) of band with the maximum value
argmin	>= 1	`propagat` (optional, default=false	(GDAL >= 3.12) Index (1-based, contrary to `numpy.argmin`) of band with the minimum value
cmul	2	•	Multiply the first band for the complex conjugate of the second
complex	2	•	Make a complex band merging two bands used as real and imag values
conj	1	•	Computes the complex conjugate of a single raster band (just a copy if the input is non-complex)
dB	1	`fact` (optional)	Perform conversion to dB of the abs of a single raster band (real or complex): `20. * log10( abs( x ) )`. The optional `fact` parameter can be set to `10` to get the alternative formula: `10. * log10( abs( x ) )`
dB2amp	1	•	Perform scale conversion from logarithmic to linear (amplitude) (i.e. `10 ^ ( x / 20 )`) of a single raster band (real only). Deprecated in GDAL 3.5. Please use the `exp` pixel function with `base = 10.` and `fact = 0.05` i.e. `1./20`
dB2pow	1	•	Perform scale conversion from logarithmic to linear (power) (i.e. `10 ^ ( x / 10 )`) of a single raster band (real only). Deprecated in GDAL 3.5. Please use the `exp` pixel function with `base = 10.` and `fact = 0.1` i.e. `1./10`
diff	2	•	Computes the difference between 2 raster bands (`b1 - b2`). Starting with GDAL 3.12, if either `b1` or `b2` is equal to the derived band's NoData value (set with <`NoDataValue`>), the result will be the NoData value.
div	2	•	Divide one raster band by another (`b1 / b2`). Starting with GDAL 3.12, if either `b1` or `b2` is equal to the derived band's NoData value (set with <`NoDataValue`>), the result will be the NoData value.

continues on next page

Table 7 – continued from previous page

PixelFunctionType	Number of input sources	PixelFunctionArguments	Description
exp	1	`base` (optional) `fact` (optional)	Computes the exponential of each element in the input band `x` (of real values): `e ^ x`. The function also accepts two optional parameters: `base` and `fact` that allow to compute the generalized formula: `base ^ ( fact * x )`. Note: this function is the recommended one to perform conversion form logarithmic scale (dB): `` 10. ^ (x / 20.)``, in this case `base = 10.` and `fact = 0.05` i.e. `1. / 20` Starting with GDAL 3.12, if `x` is equal to the derived band's NoData value (set with `<NoDataValue>`), the result will be the NoData value.
expression	1	`expressi` `dialect` (optional) `propagat` (GDAL >= 3.12, optional, default=false	Evaluate a specified expression using muparser (default) or ExprTk. The expression is specified using the "expression" argument. The dialect may be specified using the "dialect" argument. If the optional `propagateNoData` parameter is set to `true`, then if a NoData pixel is found in one of the bands, if will be propagated to the output value. Otherwise, NoData pixels will be passed to the expression as-is. The expression can then use the `NODATA` variable (or `isnodata` muparser function) to test for these pixels and handle them accordingly. Within the expression, band values can be accessed: • through the variables `B1`, `B2`, etc. • by giving a name to a source band (e.g., `<SimpleSource name="NIR">`) • or through the `BANDS` vector. With ExprTk, `BANDS` is exposed as a standard (0-indexed) vector. With muparser, it is expanded into a list of all input bands. Starting with GDAL 3.12, the variables `_CENTER_X_` and `_CENTER_Y_` can be included in the expression to access cell center coordinates. The variable `NODATA` can be included in the expression if the derived band has a NoData value. ExprTk and muparser support a number of built-in functions and control structures. Since GDAL 3.12, the function standard C++ function `fmod` is added to muparser. Refer to the documentation of those libraries for details.
geometric_mean	>= 1	`propagat` (optional, default=false	(GDAL >= 3.12) Geometric mean of input raster bands. If the optional `propagateNoData` parameter is set to `true`, then if a NoData pixel is found in one of the bands, if will be propagated to the output value. Otherwise, NoData pixels will be ignored.

continues on next page

Table 7 – continued from previous page

PixelFunctionType	Number of input sources	PixelFunctionArguments	Description
harmonic_mean	>= 1	propagat (optional, default=false propagat (optional, default=false	(GDAL >= 3.12) Harmonic mean of input raster bands. If the optional `propagateNoData` parameter is set to `true`, then if a NoData pixel is found in one of the bands, if will be propagated to the output value. Otherwise, NoData pixels will be ignored. If zero values are encountered in one of the bands, the output will be set to NoData unless `propagateZero` is set to `true`.
imag	1	•	Extract imaginary part from a single raster band (0 for non-complex)
intensity	1	•	Computes the intensity `Re( x * conj(x) )` of a single raster band (real or complex)
interpolate_exp	>= 2	`t0`, `dt`, `t`	Interpolate a value at time (or position) `t` given input sources beginning at position `t0` with spacing `dt` using exponential interpolation. Starting with GDAL 3.12, if either input source bounding `t` is equal to the NoData value of the derived band (set with <`NoDataValue`>), the result will be the NoData value.
interpolate_linear	>= 2	`t0`, `dt`, `t`	Interpolate a value at time (or position) `t` given input sources beginning at `t0` with spacing `dt` using linear interpolation. Starting with GDAL 3.12, if either input source bounding `t` is equal to the NoData value of the derived band (set with <`NoDataValue`>), the result will be the NoData value.
inv	1	`k` (optional)	Inverse (`1./x`). If the optional `k` parameter is set, then the result is multiplied by `k` (`k / x`). Starting with GDAL 3.12, if `x` is equal to the derived band's NoData value (set with <`NoDataValue`>), the result will be the NoData value.
log10	1	•	Compute the logarithm (base 10) of the abs of a single raster band (real or complex): `log10( abs( x ) )` Starting with GDAL 3.12, if `x` is equal to the derived band's NoData value (set with <`NoDataValue`>), the result will be the NoData value.
max	>= 1	propagat (optional, default=false `k`: constant (optional)	(GDAL >= 3.8) Maximum of raster band(s) and an optional constant If the optional `propagateNoData` parameter is set to `true`, then if a NoData pixel is found in one of the bands, if will be propagated to the output value. Otherwise, NoData pixels will be ignored.

continues on next page

Table 7 – continued from previous page

PixelFunctionType	Number of input sources	PixelFunctionArguments	Description
mean	>= 1	`propagat` (optional, default=false	(GDAL >= 3.12) Arithmetic mean of input raster bands. If the optional `propagateNoData` parameter is set to `true`, then if a NoData pixel is found in one of the bands, if will be propagated to the output value. Otherwise, NoData pixels will be ignored.
median	>= 1	`propagat` (optional, default=false	(GDAL >= 3.12) Median of input raster bands. If the optional `propagateNoData` parameter is set to `true`, then if a NoData pixel is found in one of the bands, if will be propagated to the output value. Otherwise, NoData pixels will be ignored.
min	>= 1	`propagat` (optional, default=false `k`: constant (optional)	(GDAL >= 3.8) Minimum of raster band(s) and an optional constant If the optional `propagateNoData` parameter is set to `true`, then if a NoData pixel is found in one of the bands, if will be propagated to the output value. Otherwise, NoData pixels will be ignored.
mod	1	•	Extract module from a single raster band (real or complex). Starting with GDAL 3.14, NoData values in real input bands will be preserved.
mode	>= 1	`propagat` (optional, default=false	(GDAL >= 3.12) Mode (most common value) of input raster bands. If the optional `propagateNoData` parameter is set to `true`, then if a NoData pixel is found in one of the bands, if will be propagated to the output value. Otherwise, NoData pixels will be ignored.
mul	>= 1	`k` (optional) `propagat` (optional, default=false	Multiply 1 or more raster bands. If the optional `k` parameter is provided then the result is multiplied by the scalar `k`. Starting with GDAL 3.12, if `propagateNoData` is true, any input pixel equal to the derived band's NoData value (set with `<NoDataValue>`) will cause the result to be the NoData value. If `propagateNoData` is false, input NoData values will be ignored.
norm_diff	2	•	Computes the normalized difference between two raster bands: `(b1 - b2)/(b1 + b2)` Starting with GDAL 3.12, if either `b1` or `b2` is equal to the derived band's NoData value (set with `<NoDataValue>`), the result will be the NoData value.
phase	1	•	Extract phase from a single raster band [-PI,PI] (0 or PI for non-complex)

continues on next page

Table 7 – continued from previous page

PixelFunctionType	Number of input sources	PixelFunctionArguments	Description
polar	2	`amplitud` (optional)	Make a complex band using input bands for amplitude and phase values `b1 * exp( j * b2 )`. The optional (string) parameter `amplitude_type` can be: • `AMPLITUDE` (default), • `INTENSITY` or • `dB`. Note: if `amplitude_type` is set to `INTENSITY` then negative values are clipped to zero.
pow	1	`power`	Raise a single raster band to a constant power, specified with argument `power` (real only) Starting with GDAL 3.12, if the input is equal to the derived band's NoData value (set with `<NoDataValue>`), the result will be the NoData value.
quantile	1	`q`	Calculate the specified quantile (`q`) of the input raster bands. If the input is equal to the derived band's NoData value (set with `<NoDataValue>`), the result will be the NoData value.
real	1	•	Extract real part from a single raster band (just a copy if the input is non-complex)
reclassify	= 1	`mapping` `default`	Reclassify values according to a mapping provided by `mapping`. The format of the mapping is `SOURCE=DEST;SOURCE=DEST;...` where each `SOURCE` element is: • either a single value • an interval (e.g., `(-inf,30]`), `NO_DATA`, or `DEFAULT`. `DEST` may be: • any constant, • `PASS_THROUGH` (to skip reclassification for certain values), • or `NO_DATA`. An error will be raised if a pixel value is not covered by any interval. A similar functionality, but with interpolation, is offered by the `ComplexSource`.
replace_nodata	= 1	`to` (optional)	Convert incoming `NoData` values to a new value, IEEE 754 *nan* by default
round	= 1	• `dig` (option	(GDAL >= 3.13) Round the input to the specified number of digits to the right of the decimal point.
scale	= 1	•	Perform scaling according to the `offset` and `scale` values of the raster band Starting with GDAL 3.12, if the input is equal to the derived band's NoData value (set with `<NoDataValue>`), the result will be the NoData value.
sqrt	1	•	Perform the square root of a single raster band (real only). Starting with GDAL 3.12, if the input is equal to the derived band's NoData value (set with `<NoDataValue>`), the result will be the NoData value.

continues on next page

Table 7 – continued from previous page

PixelFunctionType	Number of input sources	PixelFunctionArguments	Description
sum	>= 1	k (optional) propagat (optional, default=``fal	Sum 1 or more raster bands. If the optional k parameter is provided then it is added to each element of the result. Starting with GDAL 3.12, if propagateNoData is true, any input pixel equal to the derived band's NoData value (set with <NoDataValue>) will cause the result to be the NoData value. If propagateNoData is false, input NoData values will be ignored.

Example 2: VRT expression with a simple condition

This example uses a simple `if` block to calculate the derived band value depending on the value of an input band. Note the use of `>` to escape the > character in the XML attribute. Using the same VRT as *Example 1*, the following expression can be used:

```
<PixelFunctionType>expression</PixelFunctionType>
<PixelFunctionArguments dialect="muparser" expression="B1 ? 1.5*B3 : B1" />
```

or

```
<PixelFunctionType>expression</PixelFunctionType>
<PixelFunctionArguments dialect="exprtk" expression="if (B1 &gt; 1) 1.5*B3 ; else B1" />
```

Example 3: VRT expression using all bands

The `BANDS` variable provides access to all of the input bands, either individually (`BANDS[0]` for the first band) or collectively. Here, we concisely evaluate the value of a band relative to the sum of other bands. Using the same VRT as *Example 1*:

```
<PixelFunctionType>expression</PixelFunctionType>
<PixelFunctionArguments expression="B1 / sum(BANDS)" />
```

6.154.8.3 Writing Pixel Functions in C/C++

If the built-in pixel functions are not suitable, a custom function can be written in C or C++. In this case, the You could write the pixel function to compute this value and then register it with GDAL with a name such as "MyFirstFunction". To register this function with GDAL (prior to accessing any VRT datasets with derived bands that use this function), an application calls *`GDALAddDerivedBandPixelFuncWithArgs()`* with a key and a *`GDALDerivedPixelFuncWithArgs`*:

```
static const char pszMetadata[] =
"<PixelFunctionArgumentsList>"
"   <Argument name='y' description='y' type='double' mandatory='1' />"
"   <Argument type='builtin' value='offset' />"
"   <Argument type='builtin' value='scale' />"
"   <Argument type='builtin' value='NoData' />"
"   <Argument name='customConstant' type='constant' value='42'>"
"</PixelFunctionArgumentsList>";
GDALAddDerivedBandPixelFuncWithArgs("MyFirstFunction", TestFunction, pszMetadata);
```

A good time to do this is at the beginning of an application when the GDAL drivers are registered. `pszMetadata` is optional and can be `nullptr`. It can be used to declare the function signature to the user and to request additional parameters aside from the ones from the Dataset.

A *GDALDerivedPixelFuncWithArgs* is defined with a signature similar to *GDALRasterBand::IRasterIO()*:

CPLErr **TestFunction**(void **papoSources, int nSources, void *pData, int nBufXSize, int nBufYSize, *GDALDataType* eSrcType, *GDALDataType* eBufType, int nPixelSpace, int nLineSpace, *CSLConstList* papszArgs)

Parameters

- **papoSources** -- A pointer to packed rasters; one per source. The datatype of all will be the same, specified in the `eSrcType` parameter.
- **nSources** -- The number of source rasters.
- **pData** -- The buffer into which the data should be read, or from which it should be written. This buffer must contain at least `nBufXSize * nBufYSize` words of type eBufType. It is organized in left to right, top to bottom pixel order. Spacing is controlled by the `nPixelSpace` and `nLineSpace` parameters.
- **nBufXSize** -- The width of the buffer image into which the desired region is to be read, or from which it is to be written.
- **nBufYSize** -- The height of the buffer image into which the desired region is to be read, or from which it is to be written.
- **eSrcType** -- The type of the pixel values in the `papoSources` raster array.
- **eBufType** -- The type of the pixel values that the pixel function must generate in the `pData` data buffer.
- **nPixelSpace** -- The byte offset from the start of one pixel value in `pData` to the start of the next pixel value within a scanline. If defaulted (0) the size of the datatype eBufType is used.
- **nLineSpace** -- The byte offset from the start of one scanline in pData to the start of the next.
- **papszArgs** -- An optional string list of named function arguments (e.g. `y=4`)

It is also possible to register a *GDALDerivedPixelFunc* (which omits the final *CSLConstList* argument) using *GDALAddDerivedBandPixelFunc()*.

The following is an implementation of the pixel function:

```
#include "gdal.h"

CPLErr TestFunction(void **papoSources, int nSources, void *pData,
                    int nXSize, int nYSize,
                    GDALDataType eSrcType, GDALDataType eBufType,
                    int nPixelSpace, int nLineSpace,
                    CSLConstList papszArgs)
{
    int ii, iLine, iCol;
    double pix_val;
    double x0, x3, x4, x8;

    // ---- Init ----
    if (nSources != 4) return CE_Failure;

    const char *pszY = CSLFetchNameValue(papszArgs, "y");
```

(continues on next page)

(continued from previous page)

```
    if (pszY == nullptr) return CE_Failure;

    double NoData = NAN;
    const char *pszNoData = CSLFetchNameValue(papszArgs, "NoData");
    if (pszNoData != nullptr)
    {
        NoData = std::strtod(pszNoData, &end);
        if (end == pszNoData) return CE_Failure; // Could not parse
    }

    char *end = nullptr;
    double y = std::strtod(pszY, &end);
    if (end == pszY) return CE_Failure; // Could not parse

    // ---- Set pixels ----
    for( iLine = 0; iLine < nYSize; iLine++ )
    {
        for( iCol = 0; iCol < nXSize; iCol++ )
        {
            ii = iLine * nXSize + iCol;
            /* Source raster pixels may be obtained with SRCVAL macro */
            x0 = SRCVAL(papoSources[0], eSrcType, ii);
            x3 = SRCVAL(papoSources[1], eSrcType, ii);
            x4 = SRCVAL(papoSources[2], eSrcType, ii);
            x8 = SRCVAL(papoSources[3], eSrcType, ii);

            if (x0 == NoData || x3 == NoData || x4 == NoData || x8 == NoData)
                pix_val = NAN;
            else
                pix_val = sqrt((x3*x3+x4*x4)/(x0*x8)) + y;

            GDALCopyWords(&pix_val, GDT_Float64, 0,
                        ((GByte *)pData) + nLineSpace * iLine + iCol * nPixelSpace,
                        eBufType, nPixelSpace, 1);
        }
    }

    // ---- Return success ----
    return CE_None;
}
```

Setting the working data type

In addition to the subclass specification (VRTDerivedRasterBand) and the PixelFunctionType value, there is another new parameter that can come in handy: SourceTransferType. Typically the source rasters are obtained using the data type of the derived band. There might be times, however, when you want the pixel function to have access to higher resolution source data than the data type being generated. For example, you might have a derived band of type "Float", which takes a single source of type "CFloat32" or "CFloat64", and returns the imaginary portion. To accomplish this, set the SourceTransferType to "CFloat64". Otherwise the source would be converted to "Float" prior to calling the pixel function, and the imaginary portion would be lost.

```
<VRTDataset rasterXSize="1000" rasterYSize="1000">
    <VRTRasterBand dataType="Float32" band="1" subClass="VRTDerivedRasterBand">
        <Description>Magnitude</Description>
        <PixelFunctionType>MyFirstFunction</PixelFunctionType>
        <SourceTransferType>CFloat64</SourceTransferType>
        ...
```

6.154.8.4 Writing Pixel Functions in Python

In addition to pixel functions written in C/C++ as documented in the *Derived Bands (pixel functions)* section, it is possible to use pixel functions written in Python. Both CPython and NumPy are requirements at run-time.

> **Note**
>
> Python pixel functions are not enabled by default; see *Security implications*.

The subelements for VRTRasterBand (whose subclass specification must be set to VRTDerivedRasterBand) are :

- **PixelFunctionType** (required): Must be set to a function name that will be defined as a inline Python module in PixelFunctionCode element or as the form "module_name.function_name" to refer to a function in an external Python module
- **PixelFunctionLanguage** (required): Must be set to Python.
- **PixelFunctionCode** (required if PixelFunctionType is of the form "function_name", ignored otherwise). The inlined code of a Python module, that must be at least have a function whose name is given by PixelFunctionType.
- **BufferRadius** (optional, defaults to 0): Amount of extra pixels, with respect to the original RasterIO() request to satisfy, that are fetched at the left, right, bottom and top of the input and output buffers passed to the pixel function. Note that the values of the output buffer in this buffer zone will be ignored.

The signature of the Python pixel function must have the following arguments:

- **in_ar**: list of input NumPy arrays. One NumPy array for each source. If SkipNonContributingSources=true, only contributing sources will be passed.
- **out_ar**: output NumPy array to fill. The array is initialized at the right dimensions and with the VRTRasterBand.dataType.
- **xoff**: pixel offset to the top left corner of the accessed region of the band. Generally not needed except if the processing depends on the pixel position in the raster.
- **yoff** line offset to the top left corner of the accessed region of the band. Generally not needed.
- **xsize**: width of the region of the accessed region of the band. Can be used together with out_ar.shape[1] to determine the horizontal resampling ratio of the request.
- **ysize**: height of the region of the accessed region of the band. Can be used together with out_ar.shape[0] to determine the vertical resampling ratio of the request.
- **raster_xsize**: total width of the raster band. Generally not needed.
- **raster_ysize**: total height of the raster band. Generally not needed.
- **buf_radius**: radius of the buffer (in pixels) added to the left, right, top and bottom of in_ar / out_ar. This is the value of the optional BufferRadius element that can be set so that the original pixel request is extended by a given amount of pixels.
- **gt**: geotransform. Array of 6 double values.

- **kwargs**: dictionary with user arguments defined in PixelFunctionArguments

The provided `out_ar` array must be modified in-place. Any value currently returned by the pixel function is ignored.

> **Note**
>
> If wanting to fill `out_ar` from another array, use the `out_ar[:] = ...` syntax.

6.154.8.5 Python pixel function examples

Example 4: VRT that multiplies the values of the source file by a factor of 1.5

```
<VRTDataset rasterXSize="20" rasterYSize="20">
    <SRS>EPSG:26711</SRS>
    <GeoTransform>440720,60,0,3751320,0,-60</GeoTransform>
    <VRTRasterBand dataType="Byte" band="1" subClass="VRTDerivedRasterBand">
        <PixelFunctionType>multiply</PixelFunctionType>
        <PixelFunctionLanguage>Python</PixelFunctionLanguage>
        <PixelFunctionArguments factor="1.5"/>
        <PixelFunctionCode><![CDATA[
            import numpy as np
            def multiply(in_ar, out_ar, xoff, yoff, xsize, ysize, raster_xsize,
                            raster_ysize, buf_radius, gt, **kwargs):
                factor = float(kwargs['factor'])
                out_ar[:] = np.round_(np.clip(in_ar[0] * factor,0,255))
            ]]>
        </PixelFunctionCode>
        <SimpleSource>
            <SourceFilename relativeToVRT="1">byte.tif</SourceFilename>
        </SimpleSource>
    </VRTRasterBand>
</VRTDataset>
```

Example 5: VRT that adds 2 (or more) rasters

```
<VRTDataset rasterXSize="20" rasterYSize="20">
    <SRS>EPSG:26711</SRS>
    <GeoTransform>440720,60,0,3751320,0,-60</GeoTransform>
    <VRTRasterBand dataType="Byte" band="1" subClass="VRTDerivedRasterBand">
        <PixelFunctionType>add</PixelFunctionType>
        <PixelFunctionLanguage>Python</PixelFunctionLanguage>
        <PixelFunctionCode><![CDATA[
            import numpy as np
            def add(in_ar, out_ar, xoff, yoff, xsize, ysize, raster_xsize,
                            raster_ysize, buf_radius, gt, **kwargs):
                np.round_(np.clip(np.sum(in_ar, axis = 0, dtype = 'uint16'),0,255),
                        out = out_ar)
            ]]>
        </PixelFunctionCode>
        <SimpleSource>
            <SourceFilename relativeToVRT="1">byte.tif</SourceFilename>
        </SimpleSource>
```

(continues on next page)

(continued from previous page)

```
        <SimpleSource>
            <SourceFilename relativeToVRT="1">byte2.tif</SourceFilename>
        </SimpleSource>
    </VRTRasterBand>
</VRTDataset>
```

Example 6: VRT that computes hillshading using an external library

```
<VRTDataset rasterXSize="121" rasterYSize="121">
    <SRS>EPSG:4326</SRS>
    <GeoTransform>-80.004166666666663,0.008333333333333,0,
    44.004166666666663,0,-0.008333333333333</GeoTransform>
    <VRTRasterBand dataType="Byte" band="1" subClass="VRTDerivedRasterBand">
        <ColorInterp>Gray</ColorInterp>
        <SimpleSource>
            <SourceFilename relativeToVRT="1">n43.dt0</SourceFilename>
        </SimpleSource>
        <PixelFunctionLanguage>Python</PixelFunctionLanguage>
        <PixelFunctionType>hillshading.hillshade</PixelFunctionType>
        <PixelFunctionArguments scale="111120" z_factor="30" />
        <BufferRadius>1</BufferRadius>
        <SourceTransferType>Int16</SourceTransferType>
    </VRTRasterBand>
</VRTDataset>
```

with `hillshading.py`:

```
# Licence: MIT
# Copyright 2016, Even Rouault
import math

def hillshade_int(in_ar, out_ar, xoff, yoff, xsize, ysize, raster_xsize,
                        raster_ysize, radius, gt, z, scale):
    ovr_scale_x = float(out_ar.shape[1] - 2 * radius) / xsize
    ovr_scale_y = float(out_ar.shape[0] - 2 * radius) / ysize
    ewres = gt[1] / ovr_scale_x
    nsres = gt[5] / ovr_scale_y
    inv_nsres = 1.0 / nsres
    inv_ewres = 1.0 / ewres

    az = 315
    alt = 45
    degreesToRadians = math.pi / 180

    sin_alt = math.sin(alt * degreesToRadians)
    azRadians = az * degreesToRadians
    z_scale_factor = z / (8 * scale)
    cos_alt_mul_z_scale_factor = \
            math.cos(alt * degreesToRadians) * z_scale_factor
    cos_az_mul_cos_alt_mul_z_scale_factor_mul_254 = \
                254 * math.cos(azRadians) * cos_alt_mul_z_scale_factor
```

(continues on next page)

(continued from previous page)

```
    sin_az_mul_cos_alt_mul_z_scale_factor_mul_254 = \
                254 * math.sin(azRadians) * cos_alt_mul_z_scale_factor
    square_z_scale_factor = z_scale_factor * z_scale_factor
    sin_alt_mul_254 = 254.0 * sin_alt

    for j in range(radius, out_ar.shape[0]-radius):
        win_line = in_ar[0][j-radius:j+radius+1,:]
        for i in range(radius, out_ar.shape[1]-radius):
            win = win_line[:,i-radius:i+radius+1].tolist()
            x = inv_ewres * ((win[0][0] + win[1][0] + win[1][0] + win[2][0])-\
                            (win[0][2] + win[1][2] + win[1][2] + win[2][2]))
            y = inv_nsres * ((win[2][0] + win[2][1] + win[2][1] + win[2][2])-\
                            (win[0][0] + win[0][1] + win[0][1] + win[0][2]))
            xx_plus_yy = x * x + y * y
            cang_mul_254 = (sin_alt_mul_254 - \
                (y * cos_az_mul_cos_alt_mul_z_scale_factor_mul_254 - \
                    x * sin_az_mul_cos_alt_mul_z_scale_factor_mul_254)) / \
                math.sqrt(1 + square_z_scale_factor * xx_plus_yy)
            if cang_mul_254 < 0:
                out_ar[j,i] = 1
            else:
                out_ar[j,i] = 1 + round(cang_mul_254)

def hillshade(in_ar, out_ar, xoff, yoff, xsize, ysize, raster_xsize,
            raster_ysize, radius, gt, **kwargs):
    z = float(kwargs['z_factor'])
    scale= float(kwargs['scale'])
    hillshade_int(in_ar, out_ar, xoff, yoff, xsize, ysize, raster_xsize,
                raster_ysize, radius, gt, z, scale)
```

Note

When importing modules from inline Python code or when relying on out-of-line code (PixelFunctionType of the form "module_name.function_name"), you need to make sure the modules are accessible through the python path. Note that contrary to the Python interactive interpreter, the current path is not automatically added when used from GDAL. So you may need to define the **PYTHONPATH** environment variable if you get ModuleNotFoundError exceptions.

6.154.8.6 Security implications

The ability to run Python code potentially opens the door to many potential vulnerabilities if the user of GDAL may process untrusted datasets. To avoid such issues, by default, execution of Python pixel function will be disabled. The execution policy can be controlled with the following configuration options:

- **GDAL_VRT_ENABLE_PYTHON=[YES/NO/TRUSTED_MODULES]:** Defaults to `TRUSTED_MODULES`. Determine what Python code can be called from GDAL.
  - YES: all VRT scripts are considered as trusted and their Python pixel functions will be run when pixel operations are involved.
  - NO: all VRT scripts are considered untrusted, and none Python pixelfunction will be run.
  - TRUSTED_MODULES (default setting): all VRT scripts with inline Python code in their PixelFunction-

Code elements will be considered untrusted and will not be run. VRT scripts that use a PixelFunctionType of the form "module_name.function_name" will be considered as trusted, only if "module_name" is allowed in the *GDAL_VRT_PYTHON_TRUSTED_MODULES* configuration option.

- **GDAL_VRT_PYTHON_TRUSTED_MODULES=value:** The value of this configuration option is a comma separated listed of trusted module names. The '*' wildcard can be used at the name of a string to match all strings beginning with the substring before the '*' character. For example 'every*' will make 'every.thing' or 'everything' module trusted. '*' can also be used to make all modules to be trusted. The ".*" wildcard can also be used to match exact modules or submodules names. For example 'every.*' will make 'every' and 'every.thing' modules trusted, but not 'everything'.

Linking mechanism to a Python interpreter

Currently only CPython 3 is supported. The GDAL shared object is not explicitly linked at build time to any of the CPython library. When GDAL will need to run Python code, it will first determine if the Python interpreter is loaded in the current process (which is the case if the program is a Python interpreter itself, or if another program, e.g. QGIS, has already loaded the CPython library). Otherwise it will look if the *PYTHONSO* configuration option is defined. This option can be set to point to the name of the Python library to use, either as a shortname like "libpython3.10.so" if it is accessible through the Linux dynamic loader (so typically in one of the paths in /etc/ld.so.conf or LD_LIBRARY_PATH) or as a full path name like "/usr/lib/x86_64-linux-gnu/libpython3.10.so". The same holds on Windows will shortnames like "python310.dll" if accessible through the PATH or full path names like "c:\python310\python310.dll". If the *PYTHONSO* configuration option is not defined, it will look for a "python" binary in the directories of the PATH and will try to determine the related shared object (it will retry with "python3" if no "python" has been found). If the above was not successful, then a predefined list of shared objects names will be tried. As of GDAL 3.12, the order of versions searched is 3.8, 3.9, 3.10, 3.11, 3.12, 3.13, 3.14, 3.7, 3.6, 3.5. Enabling debug information (*CPL_DEBUG=ON*) will show which Python version is used.

6.154.8.7 Just-in-time compilation

The use of a just-in-time compiler may significantly speed up execution times. Numba has been successfully tested. For better performance, it is recommended to use a offline pixel function so that the just-in-time compiler may cache its compilation.

Given the following `mandelbrot.py` file :

```
# Trick for compatibility with and without numba
try:
    from numba import jit
    #print('Using numba')
    g_max_iterations = 100
except Exception:
    class jit:
        def __init__(self, nopython = True, nogil = True):
            pass

        def __call__(self, f):
            return f

    #print('Using non-JIT version')
    g_max_iterations = 25

# Use a wrapper for the entry point regarding GDAL, since GDAL cannot access
# the jit decorated function with the expected signature.
def mandelbrot(in_ar, out_ar, xoff, yoff, xsize, ysize, raster_xsize,
                        raster_ysize, r, gt, **kwargs):
```

(continues on next page)

(continued from previous page)

```
    mandelbrot_jit(out_ar, xoff, yoff, xsize, ysize, raster_xsize, raster_ysize,
g_max_iterations)

# Will make sure that the code is compiled to pure native code without Python
# fallback.
@jit(nopython=True, nogil=True, cache=True)
def mandelbrot_jit(out_ar, xoff, yoff, xsize, ysize, raster_xsize,
                        raster_ysize, max_iterations):
    ovr_factor_y = float(out_ar.shape[0]) / ysize
    ovr_factor_x = float(out_ar.shape[1]) / xsize
    for j in range( out_ar.shape[0]):
        y0 = 2.0 * (yoff + j / ovr_factor_y) / raster_ysize - 1
        for i in range(out_ar.shape[1]):
            x0 = 3.5 * (xoff + i / ovr_factor_x) / raster_xsize - 2.5
            x = 0.0
            y = 0.0
            x2 = 0.0
            y2 = 0.0
            iteration = 0
            while x2 + y2 < 4 and iteration < max_iterations:
                y = 2*x*y + y0
                x = x2 - y2 + x0
                x2 = x * x
                y2 = y * y
                iteration += 1

            out_ar[j][i] = iteration * 255 / max_iterations
```

The following VRT file can be used (to be opened with QGIS for example)

```
<VRTDataset rasterXSize="100000000" rasterYSize="100000000">
    <VRTRasterBand dataType="Byte" band="1" subClass="VRTDerivedRasterBand">
        <PixelFunctionLanguage>Python</PixelFunctionLanguage>
        <PixelFunctionType>mandelbrot.mandelbrot</PixelFunctionType>
        <Metadata>
        <MDI key="STATISTICS_MAXIMUM">255</MDI>
        <MDI key="STATISTICS_MEAN">127</MDI>
        <MDI key="STATISTICS_MINIMUM">0</MDI>
        <MDI key="STATISTICS_STDDEV">127</MDI>
        </Metadata>
        <ColorInterp>Gray</ColorInterp>
        <Histograms>
        <HistItem>
            <HistMin>-0.5</HistMin>
            <HistMax>255.5</HistMax>
            <BucketCount>256</BucketCount>
            <IncludeOutOfRange>0</IncludeOutOfRange>
            <Approximate>1</Approximate>
            <HistCounts>0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|
    0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|
    0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|
    0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|
```

(continues on next page)

(continued from previous page)

```
    0|0|0|
```

(continued from previous page)

```
        </ApproxTransformer>
        </Transformer>
        <BandList>
        <BandMapping src="1" dst="1" />
        </BandList>
    </GDALWarpOptions>
</VRTDataset>
```

6.154.10 Pansharpened VRT

A VRT can describe a dataset resulting from a pansharpening operation The pansharpening VRT combines a panchromatic band with several spectral bands of lower resolution to generate output spectral bands of the same resolution as the panchromatic band.

VRT pansharpening assumes that the panchromatic and spectral bands have the same projection (or no projection). If that is not the case, reprojection must be done in a prior step. Bands might have different geotransform matrices, in which case, by default, the resulting dataset will have as extent the union of all extents.

Currently the only supported pansharpening algorithm is a "weighted" Brovey algorithm. The general principle of this algorithm is that, after resampling the spectral bands to the resolution of the panchromatic band, a pseudo panchromatic intensity is computed from a weighted average of the spectral bands. Then the output value of the spectral band is its input value multiplied by the ratio of the real panchromatic intensity over the pseudo panchromatic intensity.

Corresponding pseudo code:

```
pseudo_panchro[pixel] = sum(weight[i] * spectral[pixel][i] for i=0 to nb_spectral_bands-
↪1)
ratio = panchro[pixel] / pseudo_panchro[pixel]
for i=0 to nb_spectral_bands-1:
    output_value[pixel][i] = input_value[pixel][i] * ratio
```

A valid pansharpened VRT must declare subClass="VRTPansharpenedDataset" as an attribute of the VRTDataset top element. The VRTDataset element must have a child **PansharpeningOptions** element. This PansharpeningOptions element must have a **PanchroBand** child element and one of several **SpectralBand** elements. PanchroBand and SpectralBand elements must have at least a **SourceFilename** child element to specify the name of the dataset. They may also have a **SourceBand** child element to specify the number of the band in the dataset (starting with 1). If not specify, the first band will be assumed. **OpenOptions** can also be specified

The SpectralBand element must generally have a **dstBand** attribute to specify the number of the output band (starting with 1) to which the input spectral band must be mapped. If the attribute is not specified, the spectral band will be taken into account in the computation of the pansharpening, but not exposed as an output band.

Panchromatic and spectral bands should generally come from different datasets, since bands of a GDAL dataset are assumed to have all the same dimensions. Spectral bands themselves can come from one or several datasets. The only constraint is that they have all the same dimensions and geotransform.

An example of a minimalist working VRT is the following. It will generates a dataset with 3 output bands corresponding to the 3 input spectral bands of multispectral.tif, pansharpened with panchromatic.tif.

```
<VRTDataset subClass="VRTPansharpenedDataset">
    <PansharpeningOptions>
        <PanchroBand>
            <SourceFilename relativeToVRT="1">panchromatic.tif</SourceFilename>
            <OpenOptions>
                <OOI key="NUM_THREADS">ALL_CPUS</OOI>
```

(continues on next page)

(continued from previous page)

```
            </OpenOptions>
            <SourceBand>1</SourceBand>
        </PanchroBand>
        <SpectralBand dstBand="1">
            <SourceFilename relativeToVRT="1">multispectral.tif</SourceFilename>
            <OpenOptions>
                <OOI key="NUM_THREADS">ALL_CPUS</OOI>
            </OpenOptions>
            <SourceBand>1</SourceBand>
        </SpectralBand>
        <SpectralBand dstBand="2">
            <SourceFilename relativeToVRT="1">multispectral.tif</SourceFilename>
            <OpenOptions>
                <OOI key="NUM_THREADS">ALL_CPUS</OOI>
            </OpenOptions>
            <SourceBand>2</SourceBand>
        </SpectralBand>
        <SpectralBand dstBand="3">
            <SourceFilename relativeToVRT="1">multispectral.tif</SourceFilename>
            <OpenOptions>
                <OOI key="NUM_THREADS">ALL_CPUS</OOI>
            </OpenOptions>
            <SourceBand>3</SourceBand>
        </SpectralBand>
    </PansharpeningOptions>
</VRTDataset>
```

In the above example, 3 output pansharpend bands will be created from the 3 declared input spectral bands. The weights will be 1/3. Cubic resampling will be used. The projection and geotransform from the panchromatic band will be reused for the VRT dataset.

It is possible to create more explicit and declarative pansharpened VRT, allowing for example to only output part of the input spectral bands (e.g. only RGB when the input multispectral dataset is RGBNir). It is also possible to add "classic" VRTRasterBands, in addition to the pansharpened bands.

In addition to the above mentioned required PanchroBand and SpectralBand elements, the PansharpeningOptions element may have the following children elements :

- **Algorithm**: to specify the pansharpening algorithm. Currently, only WeightedBrovey is supported.
- **AlgorithmOptions**: to specify the options of the pansharpening algorithm. With WeightedBrovey algorithm, the only supported option is a **Weights** child element whose content must be a comma separated list of real values assigning the weight of each of the declared input spectral bands. There must be as many values as declared input spectral bands.
- **Resampling**: the resampling kernel used to resample the spectral bands to the resolution of the panchromatic band. Can be one of Cubic (default), Average, Near, CubicSpline, Bilinear, Lanczos.
- **NumThreads**: Number of worker threads. Integer number or ALL_CPUS. If this option is not set, the *GDAL_NUM_THREADS* configuration option will be queried (its value can also be set to an integer or ALL_CPUS)
- **BitDepth**: Can be used to specify the bit depth of the panchromatic and spectral bands (e.g. 12). If not specified, the NBITS metadata item from the panchromatic band will be used if it exists.
- **NoData**: Nodata value to take into account for panchromatic and spectral bands. It will be also used as the output nodata value. If not specified and all input bands have the same nodata value, it will be implicitly used (unless the special None value is put in NoData to prevent that).

- **SpatialExtentAdjustment**: Can be one of **Union** (default), **Intersection**, **None** or **NoneWithoutWarning**. Controls the behavior when panchromatic and spectral bands have not the same geospatial extent. By default, Union will take the union of all spatial extents. Intersection the intersection of all spatial extents. None will not proceed to any adjustment at all, but will emit a warning. NoneWithoutWarning is the same as None, but in a silent way.

The below examples creates a VRT dataset with 4 bands. The first band is the panchromatic band. The 3 following bands are than red, green, blue pansharpened bands computed from a multispectral raster with red, green, blue and near-infrared bands. The near-infrared bands is taken into account for the computation of the pseudo panchromatic intensity, but not bound to an output band.

```
<VRTDataset rasterXSize="800" rasterYSize="400" subClass="VRTPansharpenedDataset">
    <SRS>WGS84</SRS>
    <GeoTransform>-180, 0.45, 0, 90, 0, -0.45</GeoTransform>
    <Metadata>
        <MDI key="DESCRIPTION">Panchromatic band + pan-sharpened red, green and blue␣
↪bands</MDI>
    </Metadata>
    <VRTRasterBand dataType="Byte" band="1" >
        <SimpleSource>
            <SourceFilename relativeToVRT="1">world_pan.tif</SourceFilename>
            <SourceBand>1</SourceBand>
        </SimpleSource>
    </VRTRasterBand>
    <VRTRasterBand dataType="Byte" band="2" subClass="VRTPansharpenedRasterBand">
        <ColorInterp>Red</ColorInterp>
    </VRTRasterBand>
    <VRTRasterBand dataType="Byte" band="3" subClass="VRTPansharpenedRasterBand">
        <ColorInterp>Green</ColorInterp>
    </VRTRasterBand>
    <VRTRasterBand dataType="Byte" band="4" subClass="VRTPansharpenedRasterBand">
        <ColorInterp>Blue</ColorInterp>
    </VRTRasterBand>
    <BlockXSize>256</BlockXSize>
    <BlockYSize>256</BlockYSize>
    <PansharpeningOptions>
        <Algorithm>WeightedBrovey</Algorithm>
        <AlgorithmOptions>
            <Weights>0.25,0.25,0.25,0.25</Weights>
        </AlgorithmOptions>
        <Resampling>Cubic</Resampling>
        <NumThreads>ALL_CPUS</NumThreads>
        <BitDepth>8</BitDepth>
        <NoData>0</NoData>
        <SpatialExtentAdjustment>Union</SpatialExtentAdjustment>
        <PanchroBand>
            <SourceFilename relativeToVRT="1">world_pan.tif</SourceFilename>
            <SourceBand>1</SourceBand>
        </PanchroBand>
        <SpectralBand dstBand="2">
            <SourceFilename relativeToVRT="1">world_rgbnir.tif</SourceFilename>
            <SourceBand>1</SourceBand>
        </SpectralBand>
        <SpectralBand dstBand="3">
```

(continues on next page)

(continued from previous page)

```
            <SourceFilename relativeToVRT="1">world_rgbnir.tif</SourceFilename>
            <SourceBand>2</SourceBand>
        </SpectralBand>
        <SpectralBand dstBand="4">
            <SourceFilename relativeToVRT="1">world_rgbnir.tif</SourceFilename>
            <SourceBand>3</SourceBand>
        </SpectralBand>
        <SpectralBand> <!-- note the absence of the dstBand attribute, to indicate
                            that the NIR band is not bound to any output band -->
            <SourceFilename relativeToVRT="1">world_rgbnir.tif</SourceFilename>
            <SourceBand>4</SourceBand>
        </SpectralBand>
    </PansharpeningOptions>
</VRTDataset>
```

6.154.11 Multidimensional VRT

Added in version 3.1.

See the dedicated *Multidimensional VRT* page.

6.154.11.1 Multidimensional VRT

Added in version 3.1.

Multidimensional VRT is a specific variant of the *VRT -- GDAL Virtual Format* format, dedicated to represent Multidimensional arrays, according to the *Multidimensional Raster Data Model*.

Here's an example of such a file:

```
<VRTDataset>
    <Group name="/">
        <Dimension name="Y" size="40"/>
        <Dimension name="X" size="30"/>

        <Array name="temperature">
            <DataType>Float64</DataType>
            <DimensionRef ref="Y"/>
            <DimensionRef ref="X"/>
            <Source>
                <SourceFilename>my.nc</SourceFilename>
                <SourceArray>temperature</SourceArray>
                <SourceSlab offset="1,1" count="10,10" step="2,1"/>
                <DestSlab offset="2,1"/>
            </Source>

            <Overviews>
                <ArrayFullName>/overview0/temperature</ArrayFullName>
            </Overviews>
        </Array>
    </Group>

    <Group name="/overview0">
        <Dimension name="Y" size="4"/>
```

(continues on next page)

(continued from previous page)

```
        <Dimension name="X" size="3"/>

        <Array name="temperature">
            <DataType>Float64</DataType>
            <DimensionRef ref="Y"/>
            <DimensionRef ref="X"/>
            <Source>
                <SourceFilename>my.nc</SourceFilename>
                <SourceArray>temperature_overview</SourceArray>
                <SourceSlab offset="1,1" count="2,2" step="2,1"/>
                <DestSlab offset="2,1"/>
            </Source>
        </Array>
    </Group>
</VRTDataset>
```

.vrt Format

A XML schema of the GDAL VRT format is available.

Virtual files stored on disk are kept in an XML format with the following elements.

VRTDataset: This is the root element for the whole GDAL dataset. It has no attributes, and must have a single Group child element with an attribute name set to "/"

```
<VRTDataset>
    <Group name="/">
```

Group: This represents a `GDALGroup`. There is at least one root group of name "/" immediately under the VRTDataset element. A Group must have a *name* attribute, and may have the following child elements, with 0:n multiplicity: Dimension, Attribute, Array, Group

Dimension: This represents a `GDALDimension`. It has the following attributes: *name* (required), *size* (required), *type* and *direction*

```
<Dimension name="X" size="30" type="HORIZONTAL_X" direction="EAST"/>
```

Attribute: This represents a `GDALAttribute`. It must have a *name* attribute and a child *DataType* element. Attribute values are stored in one or several child *Value* element(s)

The value of *DataType* may be: String, Byte, UInt16, Int16, UInt32, Int32, Float32, Float64, CInt16, CInt32, CFloat32 or CFloat64.

```
<Attribute name="foo">
    <DataType>String</DataType>
    <Value>bar</Value>
</Attribute>
```

Array: This represents a `GDALMDArray`. It must have a *name* attribute and a child *DataType* element. It may have 0 or more *DimensionRef* or *Dimension* child elements to define its dimensions. And the following elements may be optionally specified to define its properties. *SRS, *Unit*, *NoDataValue*, *Offset*, *Scale*, and optional *Overviews* To define its values, it may have one *RegularlySpacedValues* element, or zero, one or several elements among *ConstantValue*, *InlineValues*, *InlineValuesWithValueElement* or *Source*.

```
<Array name="longitude">
    <DataType>Float64</DataType>
    <DimensionRef ref="longitude"/>
    <RegularlySpacedValues start="-180" step="0.5"/>
</Array>
```

```
<Array name="time">
    <DataType>String</DataType>
    <DimensionRef ref="time"/>
    <InlineValuesWithValueElement>
        <Value>2010-01-01</Value>
        <Value>2011-01-01</Value>
        <Value>2012-01-01</Value>
    </InlineValuesWithValueElement>
</Array>
```

```
<Array name="temperature">
    <DataType>Float64</DataType>
    <DimensionRef ref="Y"/>
    <Dimension name="X" size="3"/>
    <SRS dataAxisToSRSAxisMapping="2,1">EPSG:32631</SRS>
    <Unit>Kelvin</Unit>
    <NoDataValue>-999</NoDataValue>
    <Offset>0</Offset>
    <Scale>1</Scale>
    <Source>
        <SourceFilename>my.nc</SourceFilename>
        <SourceArray>temperature</SourceArray>
    </Source>
</Array>
```

Source: This indicates that raster data should be read from a separate dataset. A Source must have a *SourceFilename*, and either a *SourceArray* (when the source is a Multidimensional dataset), or a *SourceBand* (when the source is a classic 2D dataset) child element. It may have a *SourceTranspose* child element to apply a `GDALMDArray::Transpose()` operation and a *SourceView* to apply slicing/trimming operations or extraction of a component of a compound data type (see `GDALMDArray::GetView()`). It may have a *SourceSlab* element with attributes *offset*, *count* and *step* defining respectively the starting offset of the source, the number of values along each dimension and the step between source elements. It may have a *DestSlab* element with an *offset* attribute to define where the source data is placed into the target array. SourceSlab operates on the output of SourceView if specified, which operates itself on the output of SourceTranspose if specified.

```
<Source>
    <SourceFilename>my.nc</SourceFilename>
    <SourceArray>temperature</SourceArray>
    <SourceTranspose>1,0</SourceTranspose>
    <SourceView>[...]</SourceView>
    <SourceSlab offset="1,1" count="2,2" step="2,1"/>
    <DestSlab offset="2,1"/>
</Source>
```

Overviews: An array may have zero, one or several associated reduced resolution arrays. There are two ways to define an overview array, either by a fully qualified name pointing to an array defined elsewhere in the hierarchy, or by putting its content inline. Overview arrays must have the same number of dimensions and the same data type as the full

resolution array.

```
<Array name="temperature">
    <DataType>Float64</DataType>
    <DimensionRef ref="Y"/>
    <DimensionRef ref="X"/>
    <Source>
        <SourceFilename>my.nc</SourceFilename>
        <SourceArray>temperature</SourceArray>
    </Source>

    <Overviews>
        <!-- first way: fully qualified name to another array -->
        <ArrayFullName>/temperature_level0</ArrayFullName>

        <!-- second way: inline definition of the array -->
        <Array name="temperature_level1">
            <DataType>Float64</DataType>
            <Dimension name="Y_level1"/>
            <Dimension name="X_level1"/>
            <Source>
                <SourceFilename>my.nc</SourceFilename>
                <SourceArray>temperature_level1</SourceArray>
            </Source>
        </Array>
    </Overviews>
</Array>
```

6.154.12 Processed dataset VRT

Added in version 3.9.

A VRT processed dataset is a specific variant of the *VRT -- GDAL Virtual Format* format, to apply chained processing steps that may apply to several bands at the same time.

See the dedicated *VRT processed dataset* page.

6.154.12.1 VRT processed dataset

Added in version 3.9.

A VRT processed dataset is a specific variant of the *VRT -- GDAL Virtual Format* format, to apply chained processing steps that may apply to several bands at the same time.

The following built-in algorithms are available:

- Expression: evaluate a specified expression (e.g., "(B1+B2)/B3"). Available since GDAL 3.11.
- LocalScaleOffset: apply per-pixel gain and offset coming (typically subsampled) from auxiliary datasets. Can be used for dehazing processing.
- BandAffineCombination: perform an affine transformation combination of bands.
- Trimming: apply local thresholding of saturation
- LUT: apply a look-up table (band per band)

More algorithms can be registered at run-time with the `GDALVRTRegisterProcessedDatasetFunc()` function`

Here's an example of such a file to apply various correction to a R,G,B,NIR dataset:

```
<VRTDataset subClass="VRTProcessedDataset">
  <Input>
    <SourceFilename relativeToVRT="1">source.tif</SourceFilename>
  </Input>

  <ProcessingSteps>
    <Step name="Dehazing">
      <Algorithm>LocalScaleOffset</Algorithm>

      <Argument name="relativeToVRT">true</Argument>

      <Argument name="gain_dataset_filename_1">gains.tif</Argument>
      <Argument name="gain_dataset_filename_2">gains.tif</Argument>
      <Argument name="gain_dataset_filename_3">gains.tif</Argument>
      <Argument name="gain_dataset_filename_4">gains.tif</Argument>
      <Argument name="gain_dataset_band_1">1</Argument>
      <Argument name="gain_dataset_band_2">2</Argument>
      <Argument name="gain_dataset_band_3">3</Argument>
      <Argument name="gain_dataset_band_4">4</Argument>

      <Argument name="offset_dataset_filename_1">offsets.tif</Argument>
      <Argument name="offset_dataset_filename_2">offsets.tif</Argument>
      <Argument name="offset_dataset_filename_3">offsets.tif</Argument>
      <Argument name="offset_dataset_filename_4">offsets.tif</Argument>
      <Argument name="offset_dataset_band_1">1</Argument>
      <Argument name="offset_dataset_band_2">2</Argument>
      <Argument name="offset_dataset_band_3">3</Argument>
      <Argument name="offset_dataset_band_4">4</Argument>

      <Argument name="nodata">0</Argument>
      <Argument name="min">1</Argument>
      <Argument name="max">10000</Argument>
    </Step>

    <Step name="Linear combination">
      <Algorithm>BandAffineCombination</Algorithm>
      <Argument name="coefficients_1">0,1.2,-0.2,0.0,0.0</Argument>
      <Argument name="coefficients_2">0,-0.03,1.03,0.0,0.0</Argument>
      <Argument name="coefficients_3">0,0.0,0.0,1.0,0.0</Argument>
      <Argument name="coefficients_4">0,0.0,0.0,0.0,1.0</Argument>

      <Argument name="min">1</Argument>
      <Argument name="max">10000</Argument>
    </Step>

    <Step name="Trimming">
      <Algorithm>Trimming</Algorithm>
      <Argument name="relativeToVRT">true</Argument>
      <Argument name="trimming_dataset_filename">trimming.tif</Argument>
      <Argument name="tone_ceil">10000</Argument>
      <Argument name="top_margin">0</Argument>
      <Argument name="top_rgb">10000</Argument>
    </Step>
```

(continues on next page)

(continued from previous page)

```
    <Step name="LUT">
      <Algorithm>LUT</Algorithm>
      <Argument name="lut_1">
          0:0,10000.0:255
      </Argument>
      <Argument name="lut_2">
          0:0,10000.0:255
      </Argument>
      <Argument name="lut_3">
          0:0,10000.0:255
      </Argument>
      <Argument name="lut_4">
          0:0,10000.0:255
      </Argument>
    </Step>
  </ProcessingSteps>

  <VRTRasterBand dataType="Byte" band="1" subClass="VRTProcessedRasterBand">
    <ColorInterp>Red</ColorInterp>
  </VRTRasterBand>
  <VRTRasterBand dataType="Byte" band="2" subClass="VRTProcessedRasterBand">
    <ColorInterp>Green</ColorInterp>
  </VRTRasterBand>
  <VRTRasterBand dataType="Byte" band="3" subClass="VRTProcessedRasterBand">
    <ColorInterp>Blue</ColorInterp>
  </VRTRasterBand>
  <VRTRasterBand dataType="Byte" band="4" subClass="VRTProcessedRasterBand">
  </VRTRasterBand>
</VRTDataset>
```

.vrt format

The `VRTDataset` root element must have a `subClass="VRTProcessedDataset"` attribute.

The following child elements of `VRTDataset` may be defined: `SRS`, `GeoTransform`, `Metadata`. If they are not explicitly set, they are inferred from the input dataset.

The `VRTDataset` root element must also have the 2 following child elements:

- `Input`, which must have one and only one of the following `SourceFilename` or `VRTDataset` as child elements, to define the input dataset to which to apply the processing steps. Starting with GDAL 3.11, values from the input dataset will be automatically unscaled; this can be disabled by setting the `unscale` attribute of `Input` to `false`.
- `ProcessingSteps`, with at least one child `Step` element.

Each `Step` must have a `Algorithm` child element, and an optional `name` attribute. The value of `Algorithm` must be a registered VRTProcessedDataset function. At time of writing, the following 4 algorithms are defined: `LocalScaleOffset`, `BandAffineCombination`, `Trimming` and `LUT`.

A `Step` will generally have one or several `Argument` child elements, some of them being required, others optional. Consult the documentation of each algorithm.

Starting with GDAL 3.11, a `OutputBands` element can be defined as a child element of `VRTDataset`, with the following 2 attributes:

- `count` whose value can be `FROM_SOURCE` to indicate that the output band count must be the same as the number of bands of the input dataset, `FROM_LAST_STEP` to indicate that it must be the number of output bands returned by the initialization function of the last step, or an integer value.
- `dataType` whose value can be `FROM_SOURCE` to indicate that the output band data type must be the same as the one of the input dataset, `FROM_LAST_STEP` to indicate that it must be the one returned by the initialization function of the last step, or a value among Byte, Int8, UInt16, Int16, UInt32, Int32, UInt64, Int64, Float32, Float64, CInt16, CInt32, CFloat32 or CFloat64

Example:

```
<VRTDataset subClass="VRTProcessedDataset">
  <Input>
    <SourceFilename relativeToVRT="1">source.tif</SourceFilename>
  </Input>
  <OutputBands count="FROM_LAST_STEP" dataType="FROM_LAST_STEP"/>
  <ProcessingSteps>...</ProcessingSteps>
</VRTDataset>
```

If `OutputBands` is omitted,

- if there are explicit `VRTRasterBand` elements, they must have a `subClass="VRTProcessedRasterBand"` attribute
- it there are no explicit `VRTRasterBand` elements, the number and data types of input bands are used implicitly.

Both `OutputBands` and `VRTRasterBand` elements may be defined. The information specified by `OutputBands` will be used in priority, and `VRTRasterBand` elements will be used only if they are compatible with the band count and data type specified through `OutputBands`. A situation where `OutputBands` and `VRTRasterBand` elements are both found is for example when computing statistics on a .vrt file with only `OutputBands` initially set.

LocalScaleOffset algorithm

Apply per-pixel gain and offset coming (typically subsampled) from auxiliary datasets. Can be used for dehazing processing.

The gain and offset auxiliary datasets must have a georeferencing consistent of the input dataset, but may have a different resolution.

The formula applied by that algorithm is: `output_value = clamp(input_value * gain - offset, min, max)`

The following required arguments must be specified:

- `gain_dataset_filename_{band}`: Filename to the gain dataset, where {band} must be replaced by 1 to the number of input bands.
- `gain_dataset_band_{band}`: Band number corresponding to `gain_dataset_filename_{band}`, where {band} must be replaced by 1 to the number of input bands.
- `offset_dataset_filename_{band}`: Filename to the offset dataset, where {band} must be replaced by 1 to the number of input bands.
- `offset_dataset_band_{band}`: Band number corresponding to `offset_dataset_filename_{band}`, where {band} must be replaced by 1 to the number of input bands.

The following optional arguments may be specified:

- `relativeToVRT`: Whether gain and offset filenames are relative to the VRT. Allowed values are `true` and `false`. Defaults to `false`
- `min`: Clamp minimum value, applied before writing the output value.

- `max`: Clamp maximum value, applied before writing the output value.
- `nodata`: Override the input nodata value coming from the previous step (or the input dataset for the first step).
- `gain_nodata`: Override the nodata value coming from the gain dataset(s).
- `offset_nodata`: Override the nodata value coming from the offset dataset(s).

BandAffineCombination algorithm

Perform an affine transformation combination of bands.

The following required argument must be specified:

- `coefficients_{band}`: Comma-separated coefficients for combining bands where {band} must be replaced by 1 to the number of output bands. The number of coefficients in each argument must be 1 + number_of_input_bands, where the first coefficient is a constant, the second coefficient is the weight of the first input band, the third coefficient is the weight of the second input band, etc.

The following optional arguments may be specified:

- `src_nodata`: Override the input nodata value coming from the previous step (or the input dataset for the first step).
- `dst_nodata`: Set the output nodata value.
- `replacement_nodata`: Value to substitute to a valid computed value that would be equal to dst_nodata.
- `dst_intended_datatype`: Intended datatype of output (which might be different than the working data type). Used to infer an appropriate value for replacement_nodata when it is not specified.
- `min`: Clamp minimum value, applied before writing the output value.
- `max`: Clamp maximum value, applied before writing the output value.

Trimming algorithm

Apply local thresholding of saturation, with a special processing of the R,G,B bands compared to other bands.

The pseudo algorithm used for each pixel is:

```
// Extract local saturation value from trimming image
localMaxRGB = value from TrimmingImage
reducedRGB = min ( (1-top_margin)*top_rgb/localMaxRGB ; 1)

// RGB bands specific process
RGB[] = get red, green, blue components of input buffer
maxRGB = max(RGB[])
toneMaxRGB = min ( toneCeil/maxRGB ; 1)
toneBand[] = min ( toneCeil/RGB[] ; 1)

output_value_RGB[] = min ( reducedRGB*RGB[]*toneBand[] / toneMaxRGB ; topRGB)

// Other bands processing (NIR, ...): only apply RGB reduction factor
Trimmed(OtherBands[]) = reducedRGB * OtherBands[]
```

The following required arguments must be specified:

- `trimming_dataset_filename`: Filename of the trimming dataset. It must have one single band. It must have a georeferencing consistent of the input dataset, but may have a different resolution.
- `top_rgb`: Maximum saturating RGB output value.

- `tone_ceil`: Maximum threshold beyond which we give up saturation.
- `top_margin`: Margin to allow for dynamics in brightest areas (between 0 and 1, should be close to 0)

The following optional arguments may be specified:

- `relativeToVRT`: Whether the trimming dataset filename is relative to the VRT. Allowed values are `true` and `false`. Defaults to `false`
- `red_band`: Index (one-based) of the red band. Defaults to 1.
- `green_band`: Index (one-based) of the green band. Defaults to 1.
- `blue_band`: Index (one-based) of the blue band. Defaults to 1.
- `nodata`: Override the input nodata value coming from the previous step (or the input dataset for the first step).
- `trimming_nodata`: Override the nodata value coming from the trimming dataset.

LUT

Apply a look-up table (band per band), typically to get from UInt16 to Byte data types.

The following required argument must be specified:

- `lut_{band}`: List of the form `[src value 1]:[dest value 1],[src value 2]:[dest value 2],...` .. {band} must be replaced by 1 to the number of bands.

The intermediary values are calculated using a linear interpolation between the bounding destination values of the corresponding range. Source values should be monotonically non-decreasing. Clamping is performed for input pixel values outside of the range specified by the LUT. That is, if an input pixel value is lower than the minimum source value, then the destination value corresponding to that minimum source value is used as the output pixel value. And similarly for an input pixel value that is greater than the maximum source value.

The following optional arguments may be specified:

- `src_nodata`: Override the input nodata value coming from the previous step (or the input dataset for the first step).
- `dst_nodata`: Set the output nodata value.

Expression

Evaluate an expression using muparser or, if enabled at compile-time, ExprTk.

The following argument must be specified:

- `expression`: An expression to be evaluated. Band values can be accessed by the variables `B1`, `B2`, etc. Although muparser does not support vector data types, the name `BANDS` will be expanded into a list of the available bands, allowing the use of expressions such as `sum(BANDS)`. When using ExprTk, the name `BANDS` will be a vector whose (0-indexed) elements can be accessed individually. Expressions may return more than one value; one output band will be created for each value. The expression must return the same number of values each time it is invoked.

The following optional arguments may be specified:

- `dialect`: Indicates the library that should be used to evaluate the expression. Defaults to "muparser".
- `batch_size`: When set to a value `N`, the expression will be evaluated independently in groups of up to `N` bands. The result of the expression will be the concatenation of the results from each batch.

When using ExprTk, the following configuration options are available:

- **GDAL_EXPRTK_MAX_EXPRESSION_LENGTH=value:** Defaults to `100000`. Indicates the maximum number of characters expression that will be passed to the ExprTk parser.
- **GDAL_EXPRTK_MAX_VECTOR_LENGTH=value:** Defaults to `100000`. Indicates the maximum length of a vector variables declared within an expression.
- **GDAL_EXPRTK_ENABLE_LOOPS=[YES/NO]:** Defaults to `YES`. Indicates whether looping constructs (for, while, etc.) may be used within an expression.
- **GDAL_EXPRTK_TIMEOUT_SECONDS=value:** Defaults to `1`. Indicates the maximum per-pixel runtime of an ExprTk expression. ExprTk performs runtime checks only when loops are used.

6.154.13 vrt:// connection string

Added in version 3.1.

In some contexts, it might be useful to benefit from features of VRT without having to create a file or to provide the rather verbose VRT XML content as the connection string. For that purpose, the following URI syntax is supported for the dataset name since GDAL 3.1

```
vrt://{path_to_gdal_dataset}?[bands=num1,...,numN]
```

```
vrt://{path_to_gdal_dataset}?[a_srs=srs_def]
```

```
vrt://{path_to_gdal_dataset}?[a_ullr=ulx,uly,lrx,lry]
```

For example:

```
vrt://my.tif?bands=3,2,1
```

```
vrt://my.tif?a_srs=OGC:CRS84
```

```
vrt://my.tif?a_ullr=0,1,1,-1
```

```
vrt://my.tif?bands=2&ovr=4
```

```
vrt://my.nc?sd_name=sds
```

The supported options currently are `bands`, `a_nodata`, `a_srs`, `a_ullr`, `ovr`, `expand`, `a_scale`, `a_offset`, `ot`, `gcp`, `if`, `scale`, `exponent`, `outsize`, `projwin`, `projwin_srs`, `tr`, `r`, `srcwin`, `a_gt`, `oo`, `unscale`, `a_coord_epoch`, `nogcp`, `epo`, `eco`, `sd_name`, `sd`, and `block`.

Other options may be added in the future.

The effect of the `bands` option is to change the band composition. The values specified are the source band numbers (between 1 and N), possibly out-of-order or with repetitions. The `mask` value can be used to specify the global mask band. This can also be seen as an equivalent of running *gdal_translate -of VRT -b num1 ... -b numN*.

The effect of the `a_nodata` option (added in GDAL 3.9) is to assign (override) the nodata value of the source in the same way as (*gdal_translate*).

The effect of the `a_srs` option (added in GDAL 3.7) is to assign (override) the coordinate reference system of the source in the same way as (*gdal_translate*), it may be missing, or incorrect. The value provided for `a_srs` may be a string or a file containing a srs definition.

The effect of the `a_ullr` option (added in GDAL 3.7) is to assign (override) the georeferenced bounds of the source in the same way as (*gdal_translate*). The value consists of four numeric values separated by commas, in the order 'xmin,ymax,xmax,ymin' (upper left x,y, lower right x,y).

The effect of the `ovr` option (added in GDAL 3.7) is to specify which overview level of source file must be used, with the first overview level being 0 (*gdal_translate*).

The effect of the `expand` option (added in GDAL 3.7) is to expose a dataset with 1 band with a color table as a dataset with 3 (RGB) or 4 (RGBA) bands, as with (*gdal_translate*).

The effect of the `a_scale` option (added in GDAL 3.7) is to set band scaling value (no modification of pixel values is done), as with (*gdal_translate*).

The effect of the `a_offset` option (added in GDAL 3.7) is to set band offset value (no modification of pixel values is done), as with (*gdal_translate*).

The effect of the `ot` option (added in GDAL 3.7) is to force the output image bands to have a specific data type supported by the driver as with (*gdal_translate*).

The effect of the `gcp` option (added in GDAL 3.7) is to add the indicated ground control point to the output dataset. Values are a set of numbers as per (*gdal_translate*) `pixel,line,easting,northing[,elevation]`. Multiple entries may be included. This can also be seen as an equivalent of running *gdal_translate -of VRT -gcp pixel1 line1 easting1 northing1 [elevation1] -gcp pixel2 line2 easting2 northing2 [elevation2] ... -gcp pixelN lineN eastingN northingN [elevationN]*.

The effect of the `if` option (added in GDAL 3.7) is to specify the format/driver name/s to be attempted to open the input file (*gdal_translate*). Values may be repeated separated by comma This can also be seen as an equivalent of running *gdal_translate -of VRT -if DRV1 -if DRV2 ... -if DRVN*.

The effect of the `scale` option (added in GDAL 3.7) is to rescale the input pixel values from the range **src_min** to **src_max** to the range **dst_min** to **dst_max** `src_min,src_max[,dst_min,dst_max]` either 2 or 4 comma separated values. The same rules apply for the source and destination ranges, and `scale_bn` syntax may be used as it is with (*gdal_translate*). The option `scale=true` (default if unspecified is `scale=false`) may also be used without value arguments (added in GDAL 3.8), where it results in the output range 0,255 from whatever the source range is. Do consider the need for also using `ot` option in order to accommodate the intended output range.

The effect of the `exponent` option (added in GDAL 3.7) is to apply non-linear scaling with a power function, a single value to be used with the `scale` option. The same `exponent_bn` syntax may be used in combination with `scale_bn` to target specific band/s as per (*gdal_translate*).

The effect of the `outsize` option (added in GDAL 3.7) is to set the size of the output, in numbers *pixel,line* or in fraction *pixel%,line%* as per (*gdal_translate*).

The effect of the `projwin` option (added in GDAL 3.8) is to select a subwindow from the source image in georeferenced coordinates in the same way as (*gdal_translate*). The value consists of four numeric values separated by commas, in the order 'xmin,ymax,xmax,ymin', these are in the native georeferenced coordinates of the source unless `projwin_srs` is also provided.

The effect of the `projwin_srs` option (added in GDAL 3.8) is to specify the SRS in which to interpret the coordinates given with `projwin` in the same way as (*gdal_translate*). This option only applies if `projwin` is also supplied.

The effect of the `tr` option (added in GDAL 3.8) is to set the target resolution, two positive values in georeferenced coordinates, applied in the same way as (*gdal_translate*). The value consists of two numeric values separated by commas in the order 'xres,yres'.

The effect of the `r` option (added in GDAL 3.8) is to set the resampling algorithm used, with 'nearest' as the default. This is applied in the same way as (*gdal_translate*).

The effect of the `srcwin` option (added in GDAL 3.8) is to select a subwindow from the source image based on pixel/line location as with (*gdal_translate*). The value consists of four integer values separated by commas, in the order 'xoff,yoff,xsize,ysize'.

The effect of the `a_gt` option (added in GDAL 3.8) is to override/assign the geotransform of the output as with (*gdal_translate*). The value consists of six numeric values separated by commas, in the order 'gt(0),gt(1),gt(2),gt(3),gt(4),gt(5)'.

The effect of the `oo` option (added in GDAL 3.8) is to set driver-specific dataset open options, multiple values are allowed. The value consists of string key value pairs with multiple pairs separated by commas e.g. `oo=<key>=<val>` or . `oo=<key1>=<val1>,<key2>=<val2>,...`. This is applied in the same way as (*gdal_translate*).

The effect of the `unscale` option (added in GDAL 3.8) is to apply the scale/offset metadata for the bands to convert scaled values to unscaled values. To apply this use syntax `unscale=true`, or `unscale=false` (which is the default if not specified). Do consider the need for also using `ot` option in order to accommodate the intended output range, see more details for the same argument as with (*gdal_translate*).

The effect of the `a_coord_epoch` option (added in GDAL 3.8) is to assign a coordinate epoch, linked with the output SRS as with (*gdal_translate*).

The effect of the `nogcp` option (added in GDAL 3.8) is to not copy the GCPs in the source dataset to the output dataset (*gdal_translate*). To apply this use syntax `nogcp=true`, or `nogcp=false` (which is the default if not specified).

The effect of the `epo` option (added in GDAL 3.8) is that `srcwin` or `projwin` values that fall partially outside the source raster extent will be considered as an error as per (*gdal_translate*). To apply this use syntax `epo=true`, or `epo=false` (which is the default if not specified).

The effect of the `eco` option (added in GDAL 3.8) is that `srcwin` or `projwin` values that fall completely outside the source raster extent will be considered as an error as per (*gdal_translate*). To apply this use syntax `eco=true`, or `eco=false` (which is the default if not specified).

The effect of the `sd_name` option (added in GDAL 3.9) is to choose an individual subdataset by name for sources that have multiple subdatasets. This means that rather than a fully-qualified description such as "NETCDF:myfile.nc:somearray" we may use "vrt://myfile.nc?sd_name=somearray". This option is mutually exclusive with `sd`, and with `transpose`.

The effect of the `sd` option (added in GDAL 3.9) is to choose an individual subdataset by number for sources that have multiple subdatasets. This means that rather than a fully-qualified description such as "NETCDF:myfile.nc:somearray" we may use "vrt://myfile.nc?sd=<n>" where "<n>" is between 1 and the number of subdatasets. Note that there is no guarantee of the order of the subdatasets within a source between GDAL versions (or in some cases between file series in datasets). This mode is for convenience only, please use `sd_name` to choose a subdataset by name explicitly. This option is mutually exclusive with `sd_name`, and with `transpose`.

The effect of the `transpose` option (added in GDAL 3.12) is to specify just one array by name from a multidimensional dataset and nominate the indexes for the two axes that define the 2D dataset. This is an interface to the function *`GDALMDArray::AsClassicDataset()`* in the multidimensional raster model. This can be valuable for reorienting an array that presents X and Y in YX or some other order. (There's a possible added advantage that a valid geotransform may be provided that the classic 2D model doesn't yet infer, because the multidimensional model can derive one from coordinates referenced in that form). The usage syntax is `vrt://somefile.extension?transpose=varname:iXDim,iYDim` with a no-op case `vrt://somefile.extension?transpose=varname:0,1` and `vrt://somefile.extension?transpose=varname:1,0` would be a transpose on the first two axes. There must be two unique axis indexes with values between 0 and the maximum available. This option is mutually exclusive with `sd_name` and `sd`.

The effect of the `block` option (added in GDAL 3.13) is to access a natural block from the raster as a subwindow defined by the internal block indexing as per *`GDALRasterBand::ReadBlock()`*. The value consists of two integers separated by commas, in the order 'nXBlockOff,nYBlockOff', the horizontal and vertical block offset, with 0 indicating the left-most or top-most block, 1 the next block, and so forth. Marginal blocks are extracted with reduced size if implied by imperfect divisor matching of block size versus raster size. It is an error to provide fewer or more than two values to the `block` option, and an error for values that are less than zero, or greater than the maximum block index. This option is mutually exclusive with `srcwin`, `projwin`, `outsize`, `tr`, `r`, and `ovr`.

The options may be chained together separated by '&'. (Beware the need for quoting to protect the ampersand).

```
"vrt://my.tif?a_srs=OGC:CRS84&bands=2,1&a_ullr=-180,90,180,-90"
```

6.154.14 Multi-threading optimizations

Starting with GDAL 3.6, the ComputeStatistics() implementation can benefit from multi-threading if the sources are not overlapping and belong to different datasets. This can be enabled by setting the *GDAL_NUM_THREADS* configuration option to an integer or ALL_CPUS.

Starting with GDAL 3.10, the *NUM_THREADS* open option can be set to control specifically the multi-threading of VRT datasets. It defaults to ALL_CPUS, and when set, overrides *GDAL_NUM_THREADS* or *VRT_NUM_THREADS*. It applies to ComputeStatistics() and band-level and dataset-level RasterIO(). For band-level RasterIO(), multi-threading is only available if more than 1 million pixels are requested and if the VRT is made of only non-overlapping SimpleSource or ComplexSource belonging to different datasets. For dataset-level RasterIO(), multi-threading is only available if more than 1 million pixels are requested and if the VRT is made of only non-overlapping SimpleSource belonging to different datasets.

- **NUM_THREADS=[integer/ALL_CPUS]:** Defaults to ALL_CPUS. Determines the number of threads used when an operation reads from multiple sources.

This can also be specified globally with the *VRT_NUM_THREADS* configuration option.

- **VRT_NUM_THREADS=[integer/ALL_CPUS]:** Defaults to ALL_CPUS. Determines the number of threads used when an operation reads from multiple sources.

Note that the number of threads actually used is also limited by the *GDAL_MAX_DATASET_POOL_SIZE* configuration option.

6.154.15 Performance considerations

A VRT can reference many (hundreds, thousands, or more) datasets. Due to operating system limitations, and for performance at opening time, it is not reasonable/possible to open them all at the same time. GDAL has a "pool" of datasets opened by VRT files whose maximum limit is 100 by default. When it needs to access a dataset referenced by a VRT, it checks if it is already in the pool of open datasets. If not, when the pool has reached its limit, it closes the least recently used dataset to be able to open the new one. This maximum limit of the pool can be increased by setting the *GDAL_MAX_DATASET_POOL_SIZE* configuration option to a bigger value. Note that a typical user process on Linux is limited to 1024 simultaneously opened files, and you should let some margin for shared libraries, etc... gdal_translate and gdalwarp, by default, increase the pool size to 450.

Starting with GDAL 3.7, the *GDAL_MAX_DATASET_POOL_RAM_USAGE* configuration option to a number of bytes, to limit the RAM usage of opened datasets in the pool.

6.154.16 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.155 WCS -- OGC Web Coverage Service

Driver short name

WCS

Build dependencies

libcurl

The optional GDAL WCS driver allows access to coverages served by an OGC Web Coverage Service (WCS) as raster datasets. GDAL acts as a client to the WCS server.

Accessing a WCS server involves a local service description file containing a single `<WCS_GDAL>` XML element. No additional content (including whitespace outside the root element) is allowed. Starting with GDAL 2.3, this file is typically managed automatically by the driver in a cache directory. Users are encouraged to configure its contents through driver options rather than editing it manually.

The dataset name uses the syntax `WCS:<URL>`, where `<URL>` is the WCS service endpoint. Optional parameters may be appended to specify the WCS version, coverage identifier, and other request options.

If the URL does not include a coverage name, the driver attempts to fetch the capabilities document from the server, parse it, and show the resulting metadata to the user. Available coverages are listed as subdatasets. If the URL includes a coverage name (the parameter key 'coverage' can be used irrespective of the WCS version), the driver retrieves and parses the coverage description document and creates (or updates) the service description file accordingly. A small test GetCoverage request may be issued to determine details of the served data. If the server capabilities document is not already cached, it will also be retrieved.

When using WCS version 2.0.1 (for which support is available starting at GDAL version 2.3), additional parameters can be provided to control how the coverage data model is mapped to the GDAL raster data model. If a coverage has more than two dimensions, the driver will append the coverage metadata but report zero bands. In this case, the user must provide driver options to specify how to handle the additional dimensions and data fields. See *Range and dimension subsetting*.

The WCS driver provides basic support for WCS versions 1.0.0, 1.1.0, 1.1.1, 1.1.2, and 2.0.1. Any single-file coverage in a format supported by GDAL should work. By default, the driver prefers a format containing "tiff" in its name; if none is available, it falls back to the first format offered by the server or a user can specify a preferred format using driver options. Coordinate systems are read from the DescribeCoverage response.

6.155.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.155.2 Service description file

The service description file has the following elements as immediate children of the document element. Note that when the `WCS:<URL>` syntax for a dataset name is used, the contents of the service description file is meant to be modified by using options.

- **ServiceURL**: URL of the service without parameters
- **Version**: The WCS version that is used in the communication. If the dataset name syntax `WCS:<URL>` is used the default is 2.0.1 and the server's response may change the user request, otherwise defaults to 1.0.0. Versions 1.0.0, 1.1.0, 1.1.1, 1.1.2, and 2.0.1 are supported.
- **CoverageName**: The coverage that is used for the dataset.
- **Format**: The format to use for GetCoverage calls. If not set, selected by the driver. (WCS version 2.0)
- **PreferredFormat**: The format to use for GetCoverage calls. If not set, selected by the driver. (WCS versions 1.0 and 1.1)
- **Interpolation**: The interpolation method used when scaling. Should be one of the server supported values.
- **BlockXSize**: The block width to use for block cached remote access.
- **BlockYSize**: The block height to use for block cached remote access.
- **OverviewCount**: The number of overviews to represent bands as having. Defaults to a number such that the top overview is fairly smaller (less than 1K x 1K or so).
- **NoDataValue**: The nodata value to use for all the bands (blank for none). Normally determined by the driver. With version 2.0.1 this may be a comma separated list of values, one for each band.

  Elements for controlling the range and domain:

- **Domain**: The axes that are used for the spatial dimensions. The default is to use the first two axes given by the server. The axis order swap may apply. Syntax: `axis_name,axis_name` A `field_name:field_name` in the list denotes a range of fields. (Used only with version 2.0.1)
- **DefaultTime**: A timePosition to use by default when accessing coverages with a time dimension. Populated with the last offered time position by default. (Used only with version 1.0.0)
- **FieldName**: Name of the field being accessed. Used only with version 1.1. Defaults to the first field in the DescribeCoverage. In version 1.1 the range consists of one or more fields, which may be scalar or vector. A vector field contains one or more bands.
- **BandCount**: Number of bands in the dataset, normally determined by the driver.
- **BandType**: The pixel data type to use. Normally determined by the driver.

  Elements for controlling the communication:

- **Timeout**: The timeout to use for remote service requests. If not provided, the libcurl default is used.
- **UserPwd**: May be supplied with `userid:password` to pass a userid and password to the remote server.
- **HttpAuth**: May be BASIC, NTLM or ANY to control the authentication scheme to be used.
- **GetCoverageExtra**: An additional set of keywords to add to GetCoverage requests in URL encoded form. e.g. `&RESAMPLE=BILINEAR&Band=1`. Note that the extra parameters should not be known parameters (see below).

- **DescribeCoverageExtra**: An additional set of keywords to add to DescribeCoverage requests in URL encoded form. e.g. `&CustNo=775`

  Elements that may be needed to deal with server quirks (GDAL 2.3):

  Note: The options are not propagated to the subdataset with the switch `-sd`.

- **OriginAtBoundary**: Set this flag if the server reports grid origin to be at the pixel boundary instead of the pixel center. (Use for MapServer versions <= 7.0.7 with WCS versions 1.0.0 and 2.0.1)
- **OuterExtents**: Set to consider WCS 1.1 extents as boundaries of outer pixels instead of centers of outer pixels. (Use for GeoServer).
- **BufSizeAdjust**: Set to 0.5 in WCS 1.1 if data access fails due to the response not having expected size. (Use for GeoServer).
- **OffsetsPositive**: Use with MapServer in WCS version 2.0.1 together with NrOffsets.
- **NrOffsets**: Set to 2 if the server requires that there are only two values in the GridOffsets. Use when the server is MapServer or ArcGIS. With MapServer use also OffsetsPositive.
- **GridCRSOptional**: Let the driver skip Grid* parameters from a WCS 1.1 GetCoverage request if the request is not scaled. Do not use for GeoServer.
- **NoGridAxisSwap**: Set to tell the driver not to swap axis order. When reading the grid geometry (in WCS 1.1 the origin and offsets, in WCS 2.0.1 the grid envelope, axis labels, and offsets) no axis order swap is done although it would otherwise be done if this flag is set. In 1.1 it would be done if the CRS has inverted axes. In 2.0.1 it would be done if the axisOrder of the sequenceRule in GridFunction defines so. This is needed usually both in 1.1 and 2.0.1 when parsing coverage descriptions from MapServer and GeoServer.
- **SubsetAxisSwap** Set to tell the driver to swap the axis names in boundedBy.Envelope.axisLabels when making WCS 2.0.1 GetCoverage request. Needed for GeoServer when EPSG dictates axis order swap.
- **UseScaleFactor**: Set to tell the driver to use scale by factor approach instead of scale to size when making a WCS 2.0.1 GetCoverage request. Required when the server is ArcGIS.

6.155.2.1 Range and dimension subsetting

When using WCS version 2.0.1, the range (fields/bands) and the dimension can and/or may need to be subsetted. If the data model of the coverage contains dimensions beyond the two geographic or map coordinates, those dimensions must be sliced for GDAL. The coverage may also contain a large number of fields, from which only a subset may be selected for inclusion in the GDAL dataset.

Range and dimension subsetting must be specified via URL parameters, since a single coverage can produce multiple distinct GDAL datasets. In the WCS cache, this results in separate sets of files corresponding to each GDAL dataset:

1. server Capabilities file and a GDAL dataset metadata file made from it (key = URL with WCS version number)
2. server DescribeCoverage file, a template GDAL service file made from it, and a GDAL dataset metadata file made for it (key = URL with WCS version number and coverage name)
3. the GDAL service file specifically for this dataset, and a GDAL dataset metadata file (key = URL with WCS version number, coverage name, and range and dimension subsetting parameters)

The following URL parameters are used to control the range and dimension subsetting. Note that these can also be set through options into the service file. The ones in URL take precedence.

- **RangeSubset**: Used to select a subset of coverage fields to the dataset. Syntax: `field_name, field_name:field_name,..` (Note: requires that the server implements the range subsetting extension.)
- **Subset**: Trim or slice a dimension when fetching data from the server. Syntax: *axis_name(trim_begin_value,trim_end_value);axis_name(slice_value)* Note that trimming the geographic/map coordinates is done by the driver.

6.155.2.2 Other WCS parameters

The following parameters (for WCS version 2.0.1) are supported in addition to those described above. They can be specified either via driver options or directly in the URL, with URL parameters taking precedence. Note that support for these parameters depends on the server; not all servers may recognize them.

- MediaType
- UpdateSequence
- GEOTIFF:COMPRESSION
- GEOTIFF:JPEG_QUALITY
- GEOTIFF:PREDICTOR
- GEOTIFF:INTERLEAVE
- GEOTIFF:TILING
- GEOTIFF:TILEWIDTH

6.155.2.3 Open options

When the `WCS:<URL>` dataset name syntax is used, open options are used to control the driver and the contents of the service description file. In the case the URL does not contain coverage name, the service description file is not used and thus in that case the options are not written into it. Open options are given separate to the dataset name, with GDAL utility programs they are given using the -oo switch (*-oo "NAME=VALUE"*). The -oo switch expects only one option but more options can be given repeating the switch.

In addition to DescribeCoverageExtra and GetCoverageExtra, which are stored in the service description file, there is also GetCapabilitiesExtra, which can be used as an open option when requesting the overall capabilities from the server. The open option SKIP_GETCOVERAGE can be used to prevent the driver making a GetCoverage request to the server, which it usually does if it can't determine the band count and band data type from the capabilities or coverage descriptions. This option may be needed if GetCoverage request fails.

All above listed element names can be given as options to the WCS driver. In the case of flags the option should formally be "Name=TRUE", but only "Name" suffices.

6.155.2.4 The cache

When using the `WCS:<URL>` dataset name syntax, server responses, the service description file, and metadata files are stored in a cache. If the required resources are already cached, they will be used directly, avoiding additional requests to the server.

The default location of the cache directory is `$HOME/.gdal/wcs_cache` on Linux, and `%USERPROFILE%\.gdal\wcs_cache` on Windows.

The cache contents can be seen as subdatasets using an empty URL:

```
gdalinfo "WCS:"

# or using the gdal CLI
gdal raster info "WCS:"
```

The cache control options/flags are

- **CACHE=path** Overrides the default cache location.
- **CLEAR_CACHE=YES/NO** The cache is completely initialized and all files are deleted.
- **REFRESH_CACHE=YES/NO** The cache entry, either capabilities or coverage, depending on the call at hand, is deleted.

- **DELETE_FROM_CACHE=k** The cache entry (subdataset k), is deleted.

6.155.2.5 The WCS: dataset name syntax

The URL in the dataset name is not a complete WCS request URL. The request URL, specifically, its query part, for GetCapabilities, DescribeCoverage, and GetCoverage requests is composed by the driver. Typically the user should only need to add to the server address the version and coverage parameters. The string 'coverage' can be used as the coverage parameter key although different WCS versions use different keys. 'coverage' is also always used in the cache key.

The user may add arbitrary standard and non-standard extra parameters to the URL. However, when that is done, it should be noted that the URL is a cache database key and capability documents are linked to coverage documents through the key. Please consider using the Extra open options.

6.155.2.6 Time

This driver includes experimental support for time based WCS 1.0.0 servers. On initial access the last offered time position will be identified as the DefaultTime. Each time position available for the coverage will be treated as a subdataset.

Note that time based subdatasets are not supported when the service description is the filename. Currently time support is not available for versions other than WCS 1.0.0.

6.155.2.7 Examples

Example 1: A gdalinfo call to a coverage served by MapServer

```
gdalinfo \
-oo INTERLEAVE=PIXEL \
-oo OffsetsPositive \
-oo NrOffsets=2 \
-oo NoGridAxisSwap \
-oo BandIdentifier=none \
"WCS:http://194.66.252.155/cgi-bin/BGS_EMODnet_bathymetry/ows?VERSION=2.0.1&coverage=BGS_
↪EMODNET_CentralMed-MCol"
```

Example 2: A gdal_translate call to a scaled clip of a coverage served by GeoServer

```
gdal_translate \
-oo CACHE=wcs_cache \
-oo CLEAR_CACHE \
-oo INTERLEAVE=PIXEL \
-projwin 377418 6683393.87938218 377717.879386966 6683094 \
-oo Subset="time(1985-01-01T00:00:00.000Z)" \
-outsize 60 0 \
"WCS:https://beta-karttakuva.maanmittauslaitos.fi/wcs/service/ows?version=2.0.1&
↪coverage=ortokuva__ortokuva" \
scaled.tiff
```

Example 3: List coverages available at a WCS endpoint, refreshing any cached responses

```
gdal raster info "WCS:https://geo.weather.gc.ca/geomet" --oo REFRESH_CACHE=YES
```

Example 4: Save a WCS request as a GeoTIFF

```
gdal raster convert  "WCS:https://geo.weather.gc.ca/geomet?version=2.0.1&coverage=WCPS_
↪1km_PrecipRate" \
         --oo "Subset=lat(46,48.5)" \
         --oo "Subset=lon(-90,-86)" \
         out.tif
```

Example 5: Use a pipeline to download and render data from a WCS endpoint

Bash

PowerShell

```
cat > colors.txt <<EOF
0 0 0 0
5e-08 0 0 255
1e-07 0 255 0
1e-06 255 255 0
3.8e-06 255 0 0
EOF

gdal raster pipeline \
    ! read "WCS:https://geo.weather.gc.ca/geomet?version=2.0.1&coverage=WCPS_1km_
↪PrecipRate" \
        --oo "Subset=lat(46,48.5)" \
        --oo "Subset=lon(-90,-86)" \
    ! color-map --color-map=colors.txt \
    ! write output.jpg --overwrite
```

```
$colormap = @"
0 0 0 0
5e-08 0 0 255
1e-07 0 255 0
1e-06 255 255 0
3.8e-06 255 0 0
"@

$colormap | Set-Content -Path "colors.txt"

gdal raster pipeline `
    ! read "WCS:https://geo.weather.gc.ca/geomet?version=2.0.1&coverage=WCPS_1km_
↪PrecipRate" `
        --oo "Subset=lat(46,48.5)" `
        --oo "Subset=lon(-90,-86)" `
    ! color-map --color-map=colors.txt `
    ! write output.jpg --overwrite
```

6.155.3 See Also

- OGC WCS Standards

6.156 WEBP - WEBP

Driver short name

WEBP

Build dependencies

libwebp

GDAL can read and write WebP images through the WebP library.

WebP is a new image format that provides lossy compression for photographic images. A WebP file consists of VP8 image data, and a container based on RIFF.

The driver rely on the Open Source WebP library (BSD licensed). The WebP library (at least in its version 0.1) only offers compression and decompression of whole images, so RAM might be a limitation when dealing with big images (which are limited to 16383x16383 pixels).

The WEBP driver supports 3 bands (RGB) images. It also supports 4 bands (RGBA)

XMP metadata can be extracted from the file, and will be stored as XML raw content in the xml:XMP metadata domain.

Since 3.12, if an ESRI *world file* exists with the `.webpw` or `.wld` suffixes, it will be read and used to establish the geotransform for the image.

6.156.1 Driver capabilities

Supports CreateCopy()

This driver supports the *`GDALDriver::CreateCopy()`* operation

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.156.2 Creation options

Various creation options exists, among them :

- **QUALITY=[1-100]:** Defaults to `75`. By default the quality flag is set to 75, but this option can be used to select other values. Values must be in the range 1-100. Low values result in higher compression ratios, but poorer image quality.
- **LOSSLESS=[TRUE/FALSE]:** Defaults to `FALSE`. By default, lossy compression is used. If set to TRUE, lossless compression will be used.
- **LOSSLESS_COPY=[AUTO/YES/NO]:** (GDAL >= 3.7) Defaults to `AUTO`. Whether conversion should be lossless. In AUTO or YES mode, if LOSSLESS=YES and the source dataset uses WEBP compression, lossless copying of it is done. If set to NO, or in AUTO mode if the source dataset does not use WEBP compression, the regular conversion code path is taken, resulting in a lossless or lossy copy depending on the LOSSLESS setting.

- **WORLDFILE=[YES/NO]:** (GDAL >= 3.12) Defaults to `NO`. Force the generation of an associated ESRI world file (with the extension `.wld`). See *World Files* section for details.

6.156.3 See Also

- WebP home page

6.157 WMS -- Web Map Services

Driver short name

WMS

Build dependencies

libcurl

Accessing several different types of web image services is possible using the WMS format in GDAL.

6.157.1 Driver capabilities

Supports CreateCopy()

This driver supports the *`GDALDriver::CreateCopy()`* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.157.2 XML description file

Services are accessed by creating a local service description XML file -- there are examples below for each of the supported image services. It is important that there be no spaces or other content before the `<GDAL_WMS>` element.

<GDAL_WMS>	
<Service name="WMS">	Define what mini-driver to use, currently support
<Version>1.1.1</Version>	WMS version. (optional, defaults to 1.1.1)
<ServerUrl>http://host.domain.com/wms.cgi?</ServerUrl>	WMS server URL. (required)
<SRS>EPSG:4326</SRS>	Image projection (optional, defaults to EPSG:432
<CRS>CRS:83</CRS>	Image projection (optional, defaults to EPSG:432
<ImageFormat>image/jpeg</ImageFormat>	Format in which to request data. Paletted formats

<Transparent>FALSE</Transparent>	Set to TRUE to include "transparent=TRUE" in th
<Layers>modis%2Cglobal_mosaic</Layers>	A URL encoded, comma separated string of layer
<TiledGroupName>Clementine</TiledGroupName>	Comma separated list of layers. (required for Tile
<Styles></Styles>	Comma separated list of styles. (optional)
<BBoxOrder>xyXY</BBoxOrder>	Reorder bbox coordinates arbitrarily. May be requ
</Service>	
<DataWindow>	Define size and extents of the data. (required, exc
<UpperLeftX>-180.0</UpperLeftX>	X (longitude) coordinate of upper-left corner. (op
<UpperLeftY>90.0</UpperLeftY>	Y (latitude) coordinate of upper-left corner. (optio
<LowerRightX>180.0</LowerRightX>	X (longitude) coordinate of lower-right corner. (o
<LowerRightY>-90.0</LowerRightY>	Y (latitude) coordinate of lower-right corner. (opt
<SizeX>2666666</SizeX>	Image size in pixels.
<SizeY>1333333</SizeY>	Image size in pixels.
<TileX>0</TileX>	Added to tile X value at highest resolution. (ignor
<TileY>0</TileY>	Added to tile Y value at highest resolution. (ignor
<TileLevel>0</TileLevel>	Tile level at highest resolution. (tiled image sourc
<TileCountX>0</TileCountX>	Can be used to define image size, SizeX = TileCo
<TileCountY>0</TileCountY>	Can be used to define image size, SizeY = TileCo
<YOrigin>top</YOrigin>	Can be used to define the position of the Y origin
</DataWindow>	
<Projection>EPSG:4326</Projection>	Image projection (optional, defaults to value repo
<IdentificationTolerance>2</IdentificationTolerance>	Identification tolerance (optional, defaults to 2)
<BandsCount>3</BandsCount>	Number of bands/channels, 1 for grayscale data, 3
<DataType>Byte</DataType>	Band data type, one of: Byte, Int16, UInt16, Int32
<DataValues NoData="0 0 0" min="1 1 1" max="255 255 255" />	Define NoData and/or minimum and/or maximun
<BlockSizeX>1024</BlockSizeX>	Block size in pixels. (optional, defaults to 1024, e
<BlockSizeY>1024</BlockSizeY>	Block size in pixels. (optional, defaults to 1024, e
<OverviewCount>10</OverviewCount>	Count of reduced resolution layers each having 2
<Cache>	Enable local disk cache. Allows for offline operat
<Path>./gdalwmscache</Path>	Location where to store cache files. It is safe to us
<Depth>2</Depth>	Number of directory layers. 2 will result in files b
<Extension>.jpg</Extension>	Append to cache files. (optional, defaults to none)
<Type>file</Type>	Cache type. Now supported only 'file' type. In 'file
<Expires>604800</Expires>	Time in seconds cached files will stay valid. If ca
<MaxSize>67108864</MaxSize>	The cache maximum size in bytes. If cache reach
<CleanTimeout>120</CleanTimeout>	Clean Thread Run Timeout in seconds. How ofte
<Unique>True</Unique>	If set to true the path will appended with md5 has
</Cache>	
<MaxConnections>2</MaxConnections>	Maximum number of simultaneous connections.
<Timeout>300</Timeout>	Connection timeout in seconds. (optional, default
<OfflineMode>true</OfflineMode>	Do not download any new images, use only what
<AdviseRead>true</AdviseRead>	Enable AdviseRead API call - download images i
<VerifyAdviseRead>true</VerifyAdviseRead>	Open each downloaded image and do some basic
<ClampRequests>false</ClampRequests>	Should requests, that otherwise would be partially
<UserAgent>GDAL WMS driver (https://gdal.org/frmt_wms.html)</UserAgent>	HTTP User-agent string. Some servers might requ
<Accept>mimetype>/Accept>	HTTP Accept header to specify the MIME type o
<UserPwd>user:password</UserPwd>	User and Password for HTTP authentication (opti
<UnsafeSSL>true</UnsafeSSL>	Skip SSL certificate verification. May be needed
<Referer>http://example.foo/</Referer>	HTTP Referer string. Some servers might require
<ZeroBlockHttpCodes>204,404</ZeroBlockHttpCodes>	Comma separated list of HTTP response codes th
<ZeroBlockOnServerException>true</ZeroBlockOnServerException>	Whether to treat a Service Exception returned by
</GDAL_WMS>	

Additional HTTP headers can be sent by setting the GDAL_HTTP_HEADER_FILE configuration option to point to a filename of a text file with "key: value" HTTP headers.

6.157.3 Minidrivers

The GDAL WMS driver has support for several internal 'minidrivers', which allow access to different web mapping services. Each of these services may support a different set of options in the Service block.

6.157.3.1 WMS

Communications with an OGC WMS server. Has support for both tiled and untiled requests.

WMS layers can be queried (through a GetFeatureInfo request) with the gdallocationinfo utility, or with a GetMetadataItem("Pixel_iCol_iLine", "LocationInfo") call on a band object.

```
gdallocationinfo "WMS:http://demo.opengeo.org/geoserver/gwc/service/wms?SERVICE=WMS&
↪VERSION=1.1.1&
                            REQUEST=GetMap&LAYERS=og%3Abugsites&SRS=EPSG:900913&
                            BBOX=-1.15841845090625E7,5479006.186718751,-1.
↪1505912992109375E7,5557277.703671876&
                            FORMAT=image/png&TILESIZE=256&OVERVIEWCOUNT=25&
↪MINRESOLUTION=0.0046653459640220&TILED=true"
                           -geoloc -11547071.455 5528616 -xml -b 1
```

Output:

```
Report pixel="248595" line="191985">
  <BandReport band="1">
    <LocationInfo>
      <wfs:FeatureCollection xmlns="http://www.opengis.net/wfs"
                              xmlns:wfs="http://www.opengis.net/wfs"
                              xmlns:gml="http://www.opengis.net/gml"
                              xmlns:og="http://opengeo.org"
                              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                              xsi:schemaLocation="http://opengeo.org http://demo.
↪opengeo.org/geoserver/wfs?service=WFS&version=1.0.0&request=DescribeFeatureType&
↪typeName=og%3Abugsites http://www.opengis.net/wfs http://demo.opengeo.org/geoserver/
↪schemas/wfs/1.0.0/WFS-basic.xsd">
        <gml:boundedBy>
          <gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#26713">
            <gml:coordinates xmlns:gml="http://www.opengis.net/gml" decimal="." cs=","␣
↪ts=" ">601228,4917635 601228,4917635</gml:coordinates>
          </gml:Box>
        </gml:boundedBy>
        <gml:featureMember>
          <og:bugsites fid="bugsites.40946">
            <gml:boundedBy>
              <gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#26713">
                <gml:coordinates xmlns:gml="http://www.opengis.net/gml" decimal="." cs=",
↪" ts=" ">601228,4917635 601228,4917635</gml:coordinates>
              </gml:Box>
```

(continues on next page)

(continued from previous page)

```
            </gml:boundedBy>
            <og:cat>86</og:cat>
            <og:str1>Beetle site</og:str1>
            <og:the_geom>
              <gml:Point srsName="http://www.opengis.net/gml/srs/epsg.xml#26713">
                <gml:coordinates xmlns:gml="http://www.opengis.net/gml" decimal="." cs=",
↪" ts=" ">601228,4917635</gml:coordinates>
              </gml:Point>
            </og:the_geom>
          </og:bugsites>
        </gml:featureMember>
      </wfs:FeatureCollection>
    </LocationInfo>
    <Value>255</Value>
  </BandReport>
</Report>
```

6.157.3.2 TileService

Service to support talking to a WorldWind TileService. Access is always tile based.

6.157.3.3 WorldWind

Access to web-based WorldWind tile services. Access is always tile based.

6.157.3.4 TMS

The TMS Minidriver is designed primarily to support the users of the TMS Specification. This service supports only access by tiles.

Because TMS is similar to many other 'x/y/z' flavored services on the web, this service can also be used to access these services. To use it in this fashion, you can use replacement variables, of the format ${x}, ${y}, etc.

Supported variables (name is case sensitive) are :

- ${x} -- x position of the tile
- ${y} -- y position of the tile. This can be either from the top or the bottom of the tileset, based on whether the YOrigin parameter is set to true or false.
- ${z} -- z position of the tile -- zoom level
- ${version} -- version parameter, set in the config file. Defaults to 1.0.0.
- ${format} -- format parameter, set in the config file. Defaults to 'jpg'.
- ${layer} -- layer parameter, set in the config file. Defaults to nothing.

A typical ServerURL might look like:
`http://tilecache.osgeo.org/wms-c/Basic.py/${version}/${layer}/${z}/${x}/${y}.${format}`
In order to better suit TMS users, any URL that does not contain "${" will automatically have the string above (after "Basic.py/") appended to their URL.

The TMS Service has 3 XML configuration elements that are different from other services: `Format` which defaults to `jpg`, `Layer` which has no default, and `Version` which defaults to `1.0.0`.

Additionally, the TMS service respects one additional parameter, at the DataWindow level, which is the YOrigin element. This element should be one of `bottom` (the default in TMS) or `top`, which matches OpenStreetMap and many other popular tile services.

Two examples of usage of the TMS service are included in the examples below.

6.157.3.5 OnEarth Tiled WMS

The OnEarth Tiled WMS minidriver supports the Tiled WMS specification implemented for the JPL OnEarth driver per the specification at http://web.archive.org/web/20130511182803/http://onearth.jpl.nasa.gov/tiled.html.

Only the ServerUrl and the TiledGroupName are required, most of the required information is automatically fetched from the remote server using the GetTileService method at open time.

A typical OnEarth Tiled WMS configuration file might look like:

```
<GDAL_WMS>
    <Service name="TiledWMS">
    <ServerUrl>https://gibs.earthdata.nasa.gov/twms/epsg4326/best/twms.cgi?</ServerUrl>
    <TiledGroupName>MODIS Terra CorrectedReflectance TrueColor tileset</TiledGroupName>
    <Change key="${time}">2020-02-02</Change>
    </Service>
</GDAL_WMS>
```

The TiledWMS minidriver can use the following open options :

- TiledGroupName -- The value is a string that identifies one of the tiled services available on the server
- Change -- A <Key>:<Value> pair, which will be passed to the server. The key has to match a change key that the server declares for the respective tiled group. This option can be used multiple times, for different keys. Example: - Change=time:2020-02-02

These open options are only accepted if the corresponding XML element is not present in the configuration file.

6.157.3.6 VirtualEarth

Access to web-based Virtual Earth tile services. Access is always tile based.

The ${quadkey} variable must be found in the ServerUrl element.

The DataWindow element might be omitted. The default values are :

- UpperLeftX = -20037508.34
- UpperLeftY = 20037508.34
- LowerRightX = 20037508.34
- LowerRightY = -20037508.34
- TileLevel = 21
- OverviewCount = 20
- SRS = EPSG:3857
- BlockSizeX = 256
- BlockSizeY = 256

6.157.3.7 ArcGIS REST API

Access to ArcGIS REST map service resource (untiled requests).

AGS layers can be queried (through a GetFeatureInfo request) with the gdallocationinfo utility, or with a GetMetadataItem("Pixel_iCol_iLine", "LocationInfo") call on a band object.

```
gdallocationinfo -wgs84 "<GDAL_WMS><Service name=\"AGS\"><ServerUrl>https://services.
↪arcgisonline.com/arcgis/rest/services/World_Imagery/MapServer</ServerUrl><BBoxOrder>
↪xyXY</BBoxOrder><SRS>3857</SRS></Service><DataWindow><UpperLeftX>-20037508.34</
↪UpperLeftX><UpperLeftY>20037508.34</UpperLeftY><LowerRightX>20037508.34</LowerRightX>
↪<LowerRightY>-20037508.34</LowerRightY><SizeX>512</SizeX><SizeY>512</SizeY></
↪DataWindow></GDAL_WMS>" -75.704 39.75
```

6.157.3.8 Internet Imaging Protocol (IIP)

Access to images served through IIP protocol. The server must support the JTL (Retrieve a tile as a complete JFIF image) extension of the IIP protocol.

If using the XML syntax, the ServerURL must contain the FIF parameter.

Otherwise it is also possible to use "IIP:http://foo.com/FIF=image_name" syntax as connection string, to retrieve from the server information on the full resolution dimension and the number of resolutions.

The XML definition can then be generated with "gdal_translate IIP:http://foo.com/FIF=image_name out.xml -of WMS"

6.157.3.9 International Image Interoperability Framework Image API (IIIF)

Added in version 3.12.

Access to images served through IIIF Image API 3.0.

If using the XML syntax, the ServerURL must point to the image identifier URL, such that `*url*/info.json` exists.

Otherwise it is also possible to use "IIIF:https://path/to/image/identifier" syntax as connection string, to retrieve from the server information on the full resolution dimension and the number of resolutions.

The XML definition can then be generated with "gdal_translate IIIF:https://path/to/image/identifier out.xml -of WMS"

6.157.4 Caching

Caching of remote pixel data is possible by setting a <Cache> element in the WMS (or WMTS) configuration file.

Before GDAL 3.9, if the <Path> sub-element of <Cache> was not specified, the directory of the cache was `./gdalwmscache` (that is to say a `gdalwmscache` subdirectory of the current directory), unless the *GDAL_DEFAULT_WMS_CACHE_PATH* configuration option is specified.

Starting with GDAL 3.9, the directory of the cache is set according to the following logic (first listed criterion has priority over following ones):

- Value of the <Path> sub-element of <Cache>, if specified.
- `${GDAL_DEFAULT_WMS_CACHE_PATH}` if *GDAL_DEFAULT_WMS_CACHE_PATH* is set.
- `${XDG_CACHE_HOME}/gdalwmscache` if the `XDG_CACHE_HOME` configuration option is set.
- On Unix, `${HOME}/.cache/gdalwmscache` if the `HOME` configuration option is set.
- On Windows, `${USERPROFILE}/.cache/gdalwmscache` if the `USERPROFILE` configuration option is set.
- `${CPL_TMPDIR}/gdalwmscache_${USER}` if *CPL_TMPDIR* and `USER` configuration options are set.

If CPL_TMPDIR is not set, then TMPDIR is used, or TEMP

If USER is not set, USERNAME is used if set. If neither USERNAME or USER are set, the md5sum of the filename of the configuration file is used)

- ./gdalwmscache_{md5sum(filename)} if none of the above mentioned configuration options are set.

Note that if the <Unique> element is set to true (which is its default value), a subdirectory whose name is the md5sum of the filename of the configuration file is appended to the caching directory.

The actual caching directory can be got by querying the CACHE_PATH metadata item on the dataset.

6.157.5 Configuration options

Configuration options can be specified in command-line tools using the syntax --config <NAME>=<VALUE> or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). The following configuration options are available:

- **GDAL_MAX_CONNECTIONS=<integer>:** (GDAL >= 3.2) Defaults to 2. Set the maximum number of simultaneous connections.
- **GDAL_ENABLE_WMS_CACHE=[YES/NO]:** Defaults to YES. Can be set to NO to disable caching of pixel data.
- **GDAL_DEFAULT_WMS_CACHE_PATH=<string>:** Sets the WMS/WMTS cache path is the <Cache> element is included in the WMS/WMTS configuration file without a <Path> element.

6.157.6 Examples

Warning

The below examples rely on external servers that might go down over time. They are just here to demonstrate how to use the various services.

- frmt_wms_onearth_global_mosaic.xml - Landsat mosaic from a OnEarth WMS server

```
gdal_translate -of JPEG -outsize 500 250 onearth_global_mosaic.xml onearth_global_
↪mosaic.jpg
```

```
gdal_translate -of JPEG -projwin -10 55 30 35 -outsize 500 250 onearth_global_
↪mosaic.xml onearth_global_mosaic2.jpg
```

Note : this particular server does no longer accept regular WMS queries.

- frmt_wms_metacarta_wmsc.xml - It is possible to configure a WMS Service conforming to a WMS-C cache by specifying a number of overviews and specifying the 'block size' as the tile size of the cache. The following example is a sample set up for a 19-level "Global Profile" WMS-C cache.
- frmt_wms_tileservice_bmng.xml - - TileService, Blue Marble NG (January)
- frmt_wms_tileservice_nysdop2004.xml - TileService, NYSDOP 2004
- OpenStreetMap TMS Service Example frmt_wms_openstreetmap_tms.xml: Connect to OpenStreetMap tile service. Note that this file takes advantage of the tile cache; more information about configuring the tile cache settings is available above. Please also change the <UserAgent>, to avoid the default one being used, and potentially blocked by OSM servers in case a too big usage of it would be seen.

  gdal_translate -of PNG -outsize 512 512 frmt_wms_openstreetmap_tms.xml openstreetmap.png

- MetaCarta TMS Layer Example frmt_wms_metacarta_tms.xml, accessing the default MetaCarta TMS layer. `gdal_translate -of PNG -outsize 512 256 frmt_wms_metacarta_tms.xml metacarta.png`
- BlueMarble Amazon S3 Example frmt_wms_bluemarble_s3_tms.xml accessed with the TMS minidriver.
- Google Maps frmt_wms_googlemaps_tms.xml accessed with the TMS minidriver.
- ArcGIS MapServer Tiles frmt_wms_arcgis_mapserver_tms.xml accessed with the TMS minidriver.
- OnEarth Tiled WMS Clementine frmt_twms_Clementine.xml, daily frmt_twms_daily.xml, and `STRM` frmt_twms_srtm.xml examples.
- VirtualEarth Aerial Layer frmt_wms_virtualearth.xml accessed with the VirtualEarth minidriver.
- ArcGIS online sample server layer frmt_ags_arcgisonline.xml accessed with the ArcGIS Server REST API minidriver.
- IIP online sample server layer frmt_wms_iip.xml accessed with the IIP minidriver.
- IIIF online sample server layer frmt_wms_iiif.xml accessed with the IIIF minidriver.

6.157.7 Open syntax

The WMS driver can open :

- a local service description XML file :

```
gdalinfo description_file.xml
```

- the content of a description XML file provided as filename :

```
gdalinfo "<GDAL_WMS><Service name=\"TiledWMS\"><ServerUrl>https://gibs.earthdata.
↪nasa.gov/twms/epsg4326/best/twms.cgi?</ServerUrl><TiledGroupName>MODIS Terra␣
↪CorrectedReflectance Bands367 tileset</TiledGroupName></Service></GDAL_WMS>"
```

- the base URL of a WMS service, prefixed with *WMS:* :

```
gdalinfo "WMS:http://wms.geobase.ca/wms-bin/cubeserv.cgi"
```

  A list of subdatasets will be returned, resulting from the parsing of the GetCapabilities request on that server.

  Starting with GDAL 3.10, specifying the `-if WMS` option to command line utilities accepting it, or `WMS` as the only value of the `papszAllowedDrivers` of *GDALOpenEx()*, also forces the driver to recognize the passed URL, without the `WMS:` prefix.

- a pseudo GetMap request, such as the subdataset name returned by the previous syntax :

```
gdalinfo "WMS:http://wms.geobase.ca/wms-bin/cubeserv.cgi?SERVICE=WMS&VERSION=1.1.1&
↪REQUEST=GetMap&LAYERS=DNEC_250K%3AELEVATION%2FELEVATION&SRS=EPSG:42304&BBOX=-
↪3000000,-1500000,6000000,4500000"
```

- the base URL of a Tiled WMS service, prefixed with *WMS:* and with request=GetTileService as GET argument:

```
gdalinfo "WMS:https://gibs.earthdata.nasa.gov/twms/epsg4326/best/twms.cgi?
↪request=GetTileService"
```

  A list of subdatasets will be returned, resulting from the parsing of the GetTileService request on that server.

- the URL of a REST definition for a ArcGIS MapServer:

```
gdalinfo "http://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/
↪MapServer?f=json&pretty=true"
```

- The URL of a IIP image:

```
gdalinfo "IIP:http://merovingio.c2rmf.cnrs.fr/fcgi-bin/iipsrv.fcgi?FIF=globe.
↪256x256.tif"
```

6.157.8 Generation of WMS service description XML file

The WMS service description XML file can be generated manually, or created as the output of the CreateCopy() operation of the WMS driver, only if the source dataset is itself a WMS dataset. Said otherwise, you can use gdal_translate with as source dataset any of the above syntax mentioned in "Open syntax" and as output an XML file. For example:

```
gdal_translate "http://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/
↪MapServer?f=json" wms.xml -of WMS
```

The generated file will come with default values that you may need to edit.

6.157.9 See Also

- OGC WMS Standards
- WMS Tiling Client Recommendation (WMS-C)
- WorldWind TileService
- TMS Specification
- OnEarth Tiled WMS specification
- ArcGIS Server REST API
- *WMTS -- OGC Web Map Tile Service* driver page.

6.158 WMTS -- OGC Web Map Tile Service

Driver short name

WMTS

Build dependencies

libcurl

Access to WMTS layers is possible with the GDAL WMTS client driver (needs Curl support). It support both RESTful and KVP protocols.

6.158.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.158.2 Open syntax

The WMTS driver can open :

- a local service description XML file, whose syntax is described in the below section :

```
gdalinfo gdal_wmts.xml
```

- the content of a description XML file provided as filename :

```
gdalinfo "<GDAL_WMTS><GetCapabilitiesUrl>http://maps.wien.gv.at/wmts/1.0.0/
↪WMTSCapabilities.xml</GetCapabilitiesUrl><Layer>lb</Layer></GDAL_WMTS>"
```

- a local GetCapabilities response of a WMTS service :

```
gdalinfo WMTSCapabilities.xml
```

- the URL to the GetCapabilities response of a WMTS service:

```
gdalinfo "http://maps.wien.gv.at/wmts/1.0.0/WMTSCapabilities.xml"
```

- the URL to the GetCapabilities response of a WMTS service, prefixed with *WMTS:*, and possibly with optional layer, tilematrixset, tilematrix/zoom_level, style and extendbeyonddateline parameters, with the following syntax *WMTS:url[,layer=layer_id][,tilematrixset=tms_id][,tilematrix=tm_id|,zoom_level=level][,style=style_id][,extendbeyonddateline*

```
gdalinfo "WMTS:http://maps.wien.gv.at/wmts/1.0.0/WMTSCapabilities.xml"
```

```
gdalinfo "WMTS:http://maps.wien.gv.at/wmts/1.0.0/WMTSCapabilities.xml,layer=lb"
```

- the *WMTS:* prefix with open options

```
gdalinfo WMTS: -oo URL=http://maps.wien.gv.at/wmts/1.0.0/WMTSCapabilities.xml -oo␣
↪LAYER=lb
```

In any of the above syntaxes, if several layers are present and no layer disambiguation was done with the layer parameter/open option, or if a layer has more than one style or a tile matrix set, a list of subdatasets will be returned. If there is only one layer, it will be opened on the default style and the first tile matrix set listed.

Starting with GDAL 3.10, specifying the `-if WMTS` option to command line utilities accepting it, or `WMTS` as the only value of the `papszAllowedDrivers` of *`GDALOpenEx()`*, also forces the driver to recognize the passed filename/URL, when it is not using subdataset syntax (it can typically be used to force open a HDF5 file that would be nominally recognized by the HDF5 driver).

6.158.3 Open options

The following open options are available:

- **URL=value:** URL (or filename for local files) to GetCapabilities response document. Required if not specified in the connection string (e.g if using "WMTS:" only)
- **LAYER=value:** Layer identifier
- **TILEMATRIXSET=value:** Tile Matrix Set identifier, which determines the CRS into which the layer will be exposed. Must be one of the listed tile matrix for the layer.
- **TILEMATRIX=value:** Tile Matrix identifier. Must be one of the listed tile matrix of the select tile matrix set for the layer. Mutually exclusive with ZOOM_LEVEL. If not specified the last tile matrix, i.e. the one with the best resolution, is selected.
- **ZOOM_LEVEL=value:** Index of the maximum zoom level tile matrix to use for the full resolution GDAL dataset (lower zoom levels will be used for overviews). The first one (ie the one of lower resolution) is indexed 0. Mutually exclusive with TILEMATRIX. If not specified the last tile matrix, i.e. the one with the best resolution, is selected.
- **STYLE=value:** Style identifier. Must be one of layer.
- **EXTENDBEYONDDATELINE=[YES/N]:** Whether to make the extent go over dateline and warp tile requests. See ExtendBeyondDateLine parameter of the local service description XML file described below for more details.
- **EXTENT_METHOD=[AUTO/LAYER_BBOX/TILE_MATRIX_SET/MOST_PRECISE_TILE_MATRIX]:** Defaults to `AUTO`. GDAL needs to retrieve an extent for the layer. Different sources are possible. WGS84BoundingBox element at the Layer level, BoundingBox elements with potentially several CRS at the Layer level, BoundingBox of the TileMatrixSet definitions shared by all layers, and TileMatrixLimit definitions at the Layer level. By default (AUTO), GDAL will try first with a WGS84BoundingBox/BoundingBox corresponding to the CRS implied by the select TileMatrixSet. If not available, if will fallback to a BoundingBox in another CRS and reproject it to the selected CRS. If not available, it will fallback to the most precise tile matrix of the selected TileMatrixSet and will clip it with the bounding box implied by the most precise zoom level of the TileMatrixLimit of the layer. If LAYER_BBOX is specified, only WGS84BoundingBox/BoundingBox elements are considered. If TILE_MATRIX_SET is specified, the BoundingBox element of the selected TileMatrixSet will be used. If MOST_PRECISE_TILE_MATRIX is specified, the implicit extent of the most precise tile matrix will be used.
- **CLIP_EXTENT_WITH_MOST_PRECISE_TILE_MATRIX=[YES/NO]:** (GDAL >= 3.4.2) Whether to use the implied bounds of the most precise TileMatrix to clip the layer extent (defaults to NO if the layer bounding box is used, YES otherwise)
- **CLIP_EXTENT_WITH_MOST_PRECISE_TILE_MATRIX_LIMITS=[YES/NO]:** (GDAL >= 3.4.2) Whether to use the implied bounds of the most precise TileMatrixLimit to clip the layer extent (defaults to NO if the layer bounding box is used, YES otherwise)

6.158.4 Local service description XML file

It is important that there be no spaces or other content before the `<GDAL_WMTS>` element.

<GDAL_WMTS>	
<GetCapabilitiesUrl> http://foo/WMTSCapabilities.xml </GetCapabilitiesUrl>	URL (or filename for local files) to GetCapabilities response document (required). For a KVP only server, will be like http://end_point?SERVICE=WMTS&amp ;REQUEST=GetCapabilities .

continues on next page

Table 9 – continued from previous page

<ExtraQueryParameters>foo=bar& </ExtraQueryParameters>	URL query parameters to add to all requests (GetCapabilities, GetTile, GetFeatureInfo) (added in GDAL 3.5.1)
<Layer>layer_id</Layer>	Layer identifier (optional, but may be needed to disambiguate between several layers)
<Style>style_id</Style>	Style identifier. Must be one of the listed styles for the layer. (optional, but may be needed to disambiguate between several styles)
<TileMatrixSet>tile_matrix_set_id </TileMatrixSet>	Tile Matrix Set identifier. Must be one of the listed tile matrix set for the layer. (optional, but may be needed to disambiguate between several tile matrix sets)
<TileMatrix>tile_matrix_id </TileMatrix>	Tile Matrix identifier. Must be one of the listed tile matrix of the select tile matrix set for the layer. (optional, GDAL >= 2.2. Exclusive with ZoomLevel. If not specified the last tile matrix, ie the one with the best resolution, is selected)
<ZoomLevel>int_value</ZoomLevel>	Index of the maximum zoom level / tile matrix to use. The first one (ie the one of lower resolution) is indexed 0. (optional, GDAL >= 2.2. Exclusive with TileMatrix. If not specified the last tile matrix, ie the one with the best resolution, is selected)
<Format>image/png</Format>	Tile format, used by GetTile requests. Must be one of the listed Format for the layer. (optional, but may be needed to disambiguate between several Format)
<InfoFormat>application/xml </InfoFormat>	Info format, used by GetFeatureInfo requests. Must be one of the listed InfoFormat for the layer. (optional, but may be needed to disambiguate between several InfoFormat)
<DataWindow>	Define extents of the data. (optional, when not specified the driver will query the declared extent of the layer, and if not present fallback to the extent of the select tile matrix set, taking into account potential tile matrix set limits)
<UpperLeftX>-180.0</UpperLeftX>	X (longitude/easting) coordinate of upper-left corner, in the SRS of the tile matrix set. (required if DataWindow is present)
<UpperLeftY>90.0</UpperLeftY>	Y (latitude/northing) coordinate of upper-left corner, in the SRS of the tile matrix set. (required if DataWindow is present)
<LowerRightX>180.0</LowerRightX>	X (longitude/easting) coordinate of lower-right corner, in the SRS of the tile matrix set. (required if DataWindow is present)
<LowerRightY>-90.0</LowerRightY>	Y (latitude/northing) coordinate of lower-right corner, in the SRS of the tile matrix set. (required if DataWindow is present)
</DataWindow>	
<Projection>EPSG:4326 </Projection>	Declared projection, in case the one of the TileMatrixSet is not desirable (optional, defaults to value of the TileMatrixSet)
<BandsCount>4</BandsCount>	Number of bands/channels, 1 for grayscale data, 3 for RGB, 4 for RGBA. (optional, defaults to 4)

continues on next page

Table 9 – continued from previous page

<DataType>Byte</DataType>	Band data type, one of: Byte, Int16, UInt16, Int32, UInt32, Float32, Float64, etc.. (optional, defaults to Byte)
<ExtendBeyondDateLine>false </ExtendBeyondDateLine>	Whether to make the extent go over dateline and warp tile requests. Only appropriate when the 2 following conditions are met (optional, defaults to false): • for a geodetic SRS or EPSG:3857, with tile matrix sets such as the whole [-180,180] range of longitude is entirely covered by an integral number of tiles (e.g. GoogleMapsCompatible). • AND – when the layer BoundingBox in the SRS of the tile matrix set covers the whole [-180,180] range of longitude, and that there is another BoundingBox in another SRS that is centered around longitude 180. If such alternate BoundingBox is not present in the GetCapabilities document, DataWindow must be explicitly specified – OR when the layer BoundingBox in the SRS of the tile matrix set extends beyond the dateline.
<Cache>	Enable local disk cache. Allows for offline operation. (optional, cache is disabled when absent, but it is present in the autogenerated XML, can be overridden with GDAL_ENABLE_WMS_CACHE=NO
<Path>./gdalwmscache</Path>	Location where to store cache files. It is safe to use same cache path for different data sources. (optional, defaults to ./gdalwmscache if GDAL_DEFAULT_WMS_CACHE_PATH configuration option is not specified) /vsimem/ paths are supported allowing for temporary in-memory cache
<Type>file</Type>	Cache type. Now supported only 'file' type. In 'file' cache type files are stored in file system folders. (optional, defaults to 'file')
<Depth>2</Depth>	Number of directory layers. 2 will result in files being written as cache_path/A/B/ABCDEF... (optional, defaults to 2)
<Extension>.jpg</Extension>	Append to cache files. (optional, defaults to none)
</Cache>	
<MaxConnections>2 </MaxConnections	Maximum number of simultaneous connections. (optional, defaults to 2)
<Timeout>300</Timeout>	Connection timeout in seconds. (optional, defaults to 300)
<OfflineMode>true</OfflineMode>	Do not download any new images, use only what is in cache. Useful only with cache enabled. (optional, defaults to false)
<UserAgent>GDAL WMS driver (http://gdal.org/frmt_wms.html) </UserAgent>	HTTP User-agent string. Some servers might require a well-known user-agent such as "Mozilla/5.0" (optional, defaults to "GDAL WMS driver (http://gdal.org/frmt_wms.html)

continues on next page

Table 9 – continued from previous page

<Accept>mimetype>/Accept>	HTTP Accept header to specify the MIME type of the expected output of the server. Empty by default. (added in GDAL 3.5.1)
<UserPwd>user:password</UserPwd>	User and Password for HTTP authentication (optional).
<UnsafeSSL>true</UnsafeSSL>	Skip SSL certificate verification. May be needed if server is using a self signed certificate (optional, defaults to false, but set to true in autogenerated XML).
<Referer>http://example.foo/ </Referer>	HTTP Referer string. Some servers might require it (optional).
<ZeroBlockHttpCodes>204,404 </ZeroBlockHttpCodes>	Comma separated list of HTTP response codes that will be interpreted as a 0 filled image (i.e. black for 3 bands, and transparent for 4 bands) instead of aborting the request. (optional, defaults to non set, but set to 204,404 in autogenerated XML)
<ZeroBlockOnServerException>true </ZeroBlockOnServerException>	Whether to treat a Service Exception returned by the server as a 0 filled image instead of aborting the request. (optional, defaults to false, but set to true in autogenerated XML)
</GDAL_WMTS>	

Additional HTTP headers can be sent by setting the *GDAL_HTTP_HEADER_FILE* configuration option to point to a filename of a text file with "key: value" HTTP headers.

6.158.5 GetFeatureInfo request

WMTS layers can be queried (through a GetFeatureInfo request) with the gdallocationinfo utility, or with a GetMetadataItem("Pixel_iCol_iLine", "LocationInfo") call on a band object.

```
gdallocationinfo my_wmts.xml -geoloc -11547071.455 5528616 -xml -b 1
```

6.158.6 Generation of WMTS service description XML file

The WMTS service description XML file can be generated manually, or created as the output of the CreateCopy() operation of the WMTS driver, only if the source dataset is itself a WMTS dataset. Said otherwise, you can use gdal_translate with as source dataset any of the above syntax mentioned in "Open syntax" and as output an XML file. For example:

```
gdal_translate "WMTS:http://maps.wien.gv.at/wmts/1.0.0/WMTSCapabilities.xml,layer=lb"␣
↪wmts.xml -of WMTS
```

generates the following file:

```
<GDAL_WMTS>
  <GetCapabilitiesUrl>http://maps.wien.gv.at/wmts/1.0.0/WMTSCapabilities.xml</
↪GetCapabilitiesUrl>
  <Layer>lb</Layer>
  <Style>farbe</Style>
  <TileMatrixSet>google3857</TileMatrixSet>
  <DataWindow>
    <UpperLeftX>1800035.8827671</UpperLeftX>
    <UpperLeftY>6161931.622311067</UpperLeftY>
    <LowerRightX>1845677.148953537</LowerRightX>
```

(continues on next page)

(continued from previous page)

```
    <LowerRightY>6123507.385072636</LowerRightY>
  </DataWindow>
  <BandsCount>4</BandsCount>
  <Cache />
  <UnsafeSSL>true</UnsafeSSL>
  <ZeroBlockHttpCodes>404</ZeroBlockHttpCodes>
  <ZeroBlockOnServerException>true</ZeroBlockOnServerException>
</GDAL_WMTS>
```

The generated file will come with default values that you may need to edit.

6.158.7 Caching

Caching of remote pixel data is possible by setting a <Cache> element in the WMS (or WMTS) configuration file.

Before GDAL 3.9, if the <Path> sub-element of <Cache> was not specified, the directory of the cache was `./gdalwmscache` (that is to say a `gdalwmscache` subdirectory of the current directory), unless the *GDAL_DEFAULT_WMS_CACHE_PATH* configuration option is specified.

Starting with GDAL 3.9, the directory of the cache is set according to the following logic (first listed criterion has priority over following ones):

- Value of the <Path> sub-element of <Cache>, if specified.
- `${GDAL_DEFAULT_WMS_CACHE_PATH}` if *GDAL_DEFAULT_WMS_CACHE_PATH* is set.
- `${XDG_CACHE_HOME}/gdalwmscache` if the `XDG_CACHE_HOME` configuration option is set.
- On Unix, `${HOME}/.cache/gdalwmscache` if the `HOME` configuration option is set.
- On Windows, `${USERPROFILE}/.cache/gdalwmscache` if the `USERPROFILE` configuration option is set.
- `${CPL_TMPDIR}/gdalwmscache_${USER}` if *CPL_TMPDIR* and `USER` configuration options are set.

  If `CPL_TMPDIR` is not set, then `TMPDIR` is used, or `TEMP`

  If `USER` is not set, `USERNAME` is used if set. If neither `USERNAME` or `USER` are set, the md5sum of the filename of the configuration file is used)
- `./gdalwmscache_{md5sum(filename)}` if none of the above mentioned configuration options are set.

Note that if the <Unique> element is set to true (which is its default value), a subdirectory whose name is the md5sum of the filename of the configuration file is appended to the caching directory.

The actual caching directory can be got by querying the `CACHE_PATH` metadata item on the dataset.

6.158.8 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). The following configuration options are available:

- **GDAL_MAX_CONNECTIONS=<integer>:** (GDAL >= 3.2) Defaults to `2`. Set the maximum number of simultaneous connections.
- **GDAL_ENABLE_WMS_CACHE=[YES/NO]:** Defaults to `YES`. Can be set to NO to disable caching of pixel data.

- **GDAL_DEFAULT_WMS_CACHE_PATH=<string>:** Sets the WMS/WMTS cache path is the <Cache> element is included in the WMS/WMTS configuration file without a <Path> element.

6.158.9 See Also

- OGC WMTS Standard
- *WMS -- Web Map Services* driver page.

6.159 XYZ -- ASCII Gridded XYZ

Driver short name

XYZ

Driver built-in by default

This driver is built-in by default

GDAL supports reading and writing ASCII **gridded** XYZ raster datasets (i.e. ungridded XYZ, LIDAR XYZ etc. must be opened by other means. See the documentation of the *gdal_grid* utility).

Those datasets are ASCII files with (at least) 3 columns, each line containing the X and Y coordinates of the center of the cell and the value of the cell. (Note the XYZ driver only uses the first band of the dataset. I.e., columns beyond the third are ignored.)

The spacing between each cell must be constant.

The following data organization are supported :

- Cells with same Y coordinates must be placed on consecutive lines. For a same Y coordinate value, the lines in the dataset must be organized by increasing X values. The value of the Y coordinate can increase or decrease however.
- or, starting with GDAL 3.2.1, cells with same X coordinates must be placed on consecutive lines. For a same X coordinate value, the columns must be organized by increasing or decreasing Y values. For that organization, no missing value is supported, and the whole dataset will be ingested into memory (thus the driver will limit to 100 million points).

The supported column separators are space, comma, semicolon and tabulations.

The driver tries to autodetect an header line and will look for 'x', 'lon' or 'east' names to detect the index of the X column, 'y', 'lat' or 'north' for the Y column and 'z', 'alt' or 'height' for the Z column. If no header is present or one of the column could not be identified in the header, the X, Y and Z columns (in that order) are assumed to be the first 3 columns of each line. The open option `COLUMN_ORDER` overrides these assumptions (except on 'AUTO').

The opening of a big dataset can be slow as the driver must scan the whole file to determine the dataset size and spatial resolution. The driver will autodetect the data type among Byte, Int16, Int32 or Float32.

6.159.1 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). This driver supports the following open options:

- **COLUMN_ORDER=[AUTO/XYZ/YXZ]:** (GDAL >= 3.10) Defaults to `AUTO`. Specifies the order of the columns. It overrides the header.

6.159.2 Creation options

Creation options can be specified in command-line tools using the syntax `-co <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). This driver supports the following creation options:

- **COLUMN_SEPARATOR=<string>:** String used to separate the values of the X,Y and Z columns. Defaults to one space character
- **ADD_HEADER_LINE=[YES/NO]:** Defaults to `NO`. Whether an header line must be written (content is X <col_sep> Y <col_sep> Z).
- **SIGNIFICANT_DIGITS=<integer>:** Defaults to `18`. Specifies the number of significant digits to output (%g format)
- **DECIMAL_PRECISION=<integer>:** Specifies the number of decimal places to output when writing floating-point numbers (%f format; alternative to *SIGNIFICANT_DIGITS*).

6.159.3 Driver capabilities

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.159.4 See also

- Documentation of *gdal_grid*

6.160 Zarr

Added in version 3.4.

Driver short name

Zarr

Build dependencies

Built-in by default, but liblz4, libxz (lzma), libzstd and libblosc strongly recommended to get all compressors

Zarr is a format for the storage of chunked, compressed, N-dimensional arrays. This format is supported for read and write access, and using the traditional 2D raster API or the multidimensional API

The driver supports the Zarr V2 an V3 specifications. It also supports Kerchunk reference files since GDAL 3.11.

Local and cloud storage (see *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*) are supported in read and write.

6.160.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports CreateCopy()

This driver supports the *GDALDriver::CreateCopy()* operation

Supports Georeferencing

This driver supports georeferencing

Supports multidimensional API

This driver supports the *Multidimensional Raster Data Model*

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

6.160.2 Concepts

A Zarr dataset is made of a hierarchy of nodes, with intermediate nodes being groups (*GDALGroup*), and leafs being arrays (*GDALMDArray*).

6.160.3 Dataset name

For Zarr V2, the dataset name recognized by the Open() method of the driver is a directory that contains a `.zgroup` file, a `.zarray` file or a `.zmetadata` file (consolidated metadata). For faster exploration, the driver will use consolidated metadata by default when found.

For Zarr V3, the dataset name recognized by the Open() method of the driver is a directory that contains a `zarr.json` file (root of the dataset).

For datasets on file systems where file listing is not reliable, as often with /vsicurl/, it is also possible to prefix the directory name with `ZARR:`, and it is necessary to surround the /vsicurl/-prefixed URL with double quotes. e.g *ZARR:"/vsicurl/https://example.org/foo.zarr"*. Note that when passing such string in a command line shell, extra quoting might be necessary to preserve the double-quoting.

For example with a Bash shell, the whole connection string needs to be surrounded with single-quote characters:

```
gdalmdiminfo 'ZARR:"/vsicurl/https://example.org/foo.zarr"'
```

6.160.4 Zarr stores without directory listing

Added in version 3.12.

Sometimes remote Zarr stores don't have a reliable directory listing. In such cases, one can point to one of the following metadata files for GDAL to detect the correct driver to open the Zarr store:

- `zarr.json`
- `.zmetadata`
- `.zgroup`
- `.zarray`

6.160.5 Sharding support

Added in version 3.13.

Zarr v3 sharding is supported since GDAL 3.13. Write support involves setting the *SHARD_CHUNK_SHAPE* array creation option.

6.160.6 Multiscales (overviews / pyramids)

Added in version 3.13.

The driver supports reading the Zarr multiscales convention for Zarr V3 datasets. This convention describes a pyramid of arrays at decreasing resolutions within a group hierarchy.

When a Zarr V3 array has a parent (or grandparent) group whose attributes contain a `zarr_conventions` entry with the multiscales UUID and a `multiscales` attribute with a `layout` array, the driver exposes lower-resolution levels as overviews via *GDALMDArray::GetOverview()* and the classic raster band overview API.

Overviews can be generated using *GDALMDArray::BuildOverviews()* or equivalently via *GDALDataset::BuildOverviews()* on datasets obtained through *GDALMDArray::AsClassicDataset()*. For arrays with more than two dimensions, only the spatial dimensions are downsampled; non-spatial dimensions (e.g., time) are preserved. Each overview level is resampled sequentially from the previous level (e.g., 4x from 2x, not from base). Codec settings are inherited from the source array. Calling `BuildOverviews` replaces all existing overviews (unlike the default `GDALDataset::BuildOverviews` behavior which adds new levels).

6.160.7 Kerchunk reference stores

Added in version 3.11.

The driver supports reading "virtual" Zarr datasets where the Zarr logical hierarchy does not directly translate into a hierarchy of files on the filesystem, but through a file that contains a store of references to chunk locations (possibly remote). Such stores are generated by the Python Kerchunk library.

There are 2 types of Kerchunk reference stores:

- JSON reference stores, where the Zarr dataset is entirely described in a single JSON file. This comes in two versions: Version 0 and Version 1.

  Such stores can be opened with:

  ```
  gdalmdiminfo "/vsikerchunk_json_ref//path/to/reference.json"
  ```

  or sometimes, when the header of the file is sufficient to be recognized as a Kerchunk reference store, with just:

  ```
  gdalmdiminfo "/path/to/reference.json"
  ```

  Note that the "templates" and "gen" features of Version 1 are not supported.

- Parquet reference stores, where the Zarr dataset is described in a `.zmetadata` file that contains only the JSON definitions of the groups and arrays. The chunks themselves are cataloged in Parquet files in subdirectories: https://fsspec.github.io/kerchunk/spec.html#parquet-references

  Such stores can be opened with:

  ```
  gdalmdiminfo 'ZARR:"/path/to/directory/where/.zmetadata/is/located"'
  ```

  That is exactly like a regular Zarr dataset.

  To be able to read pixel values, the *(Geo)Parquet* driver must be available.

As JSON reference stores can be very large and slow to parse, the *CACHE_KERCHUNK_JSON* open option can be set to `YES` to ask to generate and use a local Parquet reference store that is cached in `$HOME/.gdal/zarr_kerchunk_cache`. This implies the *(Geo)Parquet* driver is available.

The driver does not rotate cached stores in the local cache. It is the responsibility of the user to manage its content and remove obsolete datasets.

It is also possible to convert JSON reference store into a Parquet one using the *CONVERT_TO_KERCHUNK_PARQUET_REFERENCE* creation option set to `YES`.

For example:

```
gdal_translate -of ZARR -co CONVERT_TO_KERCHUNK_PARQUET_REFERENCE=YES store.json store.
↪parq
```

6.160.8 Compression methods

Compression methods available depend on how GDAL is built, and libblosc too.

A full-feature build will show:

```
$ gdalinfo --format Zarr

[...]

  Other metadata items:
    COMPRESSORS=blosc,zlib,gzip,lzma,zstd,lz4
    BLOSC_COMPRESSORS=blosclz,lz4,lz4hc,snappy,zlib,zstd
```

For specific uses, it is also possible to register at run-time extra compressors and decompressors with *CPLRegisterCompressor()* and *CPLRegisterDecompressor()*.

6.160.9 XArray _ARRAY_DIMENSIONS

The driver support the `_ARRAY_DIMENSIONS` special attribute used by XArray to store the dimension names of an array.

6.160.10 NCZarr extensions

The driver support the NCZarr v2 extensions of storing the dimension names of an array (read-only)

6.160.11 Georeferencing encoding (CRS and geotransformation matrix)

The Zarr specification has no provision for spatial reference system encoding. Several conventions

6.160.11.1 GDAL convention

Before GDAL 3.13, the only convention supported both in reading and writing was the `GDAL` one, using a `_CRS` attribute. The geotransformation matrix, when no rotation terms is present, is encoded as `X` and `Y` one-dimensional coordinate arrays.

The `_CRS` attribute is a dictionary that may contain one or several of the following keys: `url` (using a OGC CRS URL), `wkt` (WKT:2019 used by default on writing, WKT1 also supported on reading.), `projjson`. On reading, it will use `url` by default, if not found will fallback to `wkt` and then `projjson`.

Example:

```
{
  "_CRS":{
    "wkt":"PROJCRS[\"NAD27 \/ UTM zone 11N\",BASEGEOGCRS[\"NAD27\",DATUM[\"North␣
↪American Datum 1927\",ELLIPSOID[\"Clarke 1866\",6378206.4,294.978698213898,LENGTHUNIT[\
↪"metre\",1]]],PRIMEM[\"Greenwich\",0,ANGLEUNIT[\"degree\",0.0174532925199433]],ID[\
↪"EPSG\",4267]],CONVERSION[\"UTM zone 11N\",METHOD[\"Transverse Mercator\",ID[\"EPSG\",
↪9807]],PARAMETER[\"Latitude of natural origin\",0,ANGLEUNIT[\"degree\",0.
↪0174532925199433],ID[\"EPSG\",8801]],PARAMETER[\"Longitude of natural origin\",-117,
↪ANGLEUNIT[\"degree\",0.0174532925199433],ID[\"EPSG\",8802]],PARAMETER[\"Scale factor␣
↪at natural origin\",0.9996,SCALEUNIT[\"unity\",1],ID[\"EPSG\",8805]],PARAMETER[\"False␣
↪easting\",500000,LENGTHUNIT[\"metre\",1],ID[\"EPSG\",8806]],PARAMETER[\"False northing\
↪",0,LENGTHUNIT[\"metre\",1],ID[\"EPSG\",8807]]],CS[Cartesian,2],AXIS[\"easting\",east,
↪ORDER[1],LENGTHUNIT[\"metre\",1]],AXIS[\"northing\",north,ORDER[2],LENGTHUNIT[\"metre\
↪",1]],ID[\"EPSG\",26711]]",

    "projjson":{
      "$schema":"https:\/\/proj.org\/schemas\/v0.2\/projjson.schema.json",
      "type":"ProjectedCRS",
      "name":"NAD27 \/ UTM zone 11N",
      "base_crs":{
        "name":"NAD27",
        "datum":{
          "type":"GeodeticReferenceFrame",
          "name":"North American Datum 1927",
          "ellipsoid":{
            "name":"Clarke 1866",
            "semi_major_axis":6378206.4,
            "inverse_flattening":294.978698213898
          }
        },
        "coordinate_system":{
```

(continues on next page)

(continued from previous page)

```
          "subtype":"ellipsoidal",
          "axis":[
            {
              "name":"Geodetic latitude",
              "abbreviation":"Lat",
              "direction":"north",
              "unit":"degree"
            },
            {
              "name":"Geodetic longitude",
              "abbreviation":"Lon",
              "direction":"east",
              "unit":"degree"
            }
          ]
        },
        "id":{
          "authority":"EPSG",
          "code":4267
        }
      },
      "conversion":{
        "name":"UTM zone 11N",
        "method":{
          "name":"Transverse Mercator",
          "id":{
            "authority":"EPSG",
            "code":9807
          }
        },
        "parameters":[
          {
            "name":"Latitude of natural origin",
            "value":0,
            "unit":"degree",
            "id":{
              "authority":"EPSG",
              "code":8801
            }
          },
          {
            "name":"Longitude of natural origin",
            "value":-117,
            "unit":"degree",
            "id":{
              "authority":"EPSG",
              "code":8802
            }
          },
          {
            "name":"Scale factor at natural origin",
            "value":0.9996,
```

(continues on next page)

(continued from previous page)


```
            "unit":"unity",
            "id":{
              "authority":"EPSG",
              "code":8805
            }
          },
          {
            "name":"False easting",
            "value":500000,
            "unit":"metre",
            "id":{
              "authority":"EPSG",
              "code":8806
            }
          },
          {
            "name":"False northing",
            "value":0,
            "unit":"metre",
            "id":{
              "authority":"EPSG",
              "code":8807
            }
          }
        ]
      },
      "coordinate_system":{
        "subtype":"Cartesian",
        "axis":[
          {
            "name":"Easting",
            "abbreviation":"",
            "direction":"east",
            "unit":"metre"
          },
          {
            "name":"Northing",
            "abbreviation":"",
            "direction":"north",
            "unit":"metre"
          }
        ]
      },
      "id":{
        "authority":"EPSG",
        "code":26711
      }
    },

    "url":"http:\/\/www.opengis.net\/def\/crs\/EPSG\/0\/26711"
  }
}
```

6.160.11.2 SPATIAL_PROJ convention

Added in version 3.13.

Since GDAL 3.13, the Zarr spatial and geo-proj conventions are supported in reading, and in writing when the `GEOREFERENCING_CONVENTION` creation option is set to `SPATIAL_PROJ`. X and Y coordinate arrays are written only if the geotransformation matrix has no rotation terms.

Example:

```
{
    "attributes": {
        "proj:code": "EPSG:26711",
        "spatial:bbox": [
            440720.0,
            3750120.0,
            441920.0,
            3751320.0,
        ],
        "spatial:transform_type": "affine",
        "spatial:transform": [
            60.0,
            0.0,
            440720.0,
            0.0,
            -60.0,
            3751320.0,
        ],
        "spatial:registration": "pixel",
        "spatial:dimensions": ["Y", "X"],
        "zarr_conventions": [
            {
                "schema_url": "https://raw.githubusercontent.com/zarr-experimental/geo-
↪proj/refs/tags/v1/schema.json",
                "spec_url": "https://github.com/zarr-experimental/geo-proj/blob/v1/
↪README.md",
                "uuid": "f17cb550-5864-4468-aeb7-f3180cfb622f",
                "name": "proj:",
                "description": "Coordinate reference system information for geospatial␣
↪data",
            },
            {
                "schema_url": "https://raw.githubusercontent.com/zarr-conventions/
↪spatial/refs/tags/v1/schema.json",
                "spec_url": "https://github.com/zarr-conventions/spatial/blob/v1/README.
↪md",
                "uuid": "689b58e2-cf7b-45e0-9fff-9cfc0883d6b4",
                "name": "spatial:",
                "description": "Spatial coordinate information",
            },
        ]
    }
}
```

6.160.11.3 netCDF CF conventions

Added in version 3.9.

The driver supports reading a CRS using the CF conventions.

6.160.12 Particularities of the classic raster API

If the Zarr dataset contains one single array with 2 dimensions, it will be exposed as a regular GDALDataset when using the classic raster API. If the dataset contains more than one such single array, or arrays with 3 or more dimensions, the driver will list subdatasets to access each array and/or 2D slices within arrays with 3 or more dimensions.

6.160.13 Open options

The following dataset open options are available:

- **LIST_ALL_ARRAYS=[YES/NO]:** (GDAL >= 3.11) Defaults to `NO`. In classic 2D mode, whereas the subdataset list should include all arrays, including those with 0 or 1 dimension.
- **USE_CONSOLIDATED_METADATA=[YES/NO]:** Defaults to `YES`. Whether to use consolidated metadata (from `.zmetadata` for Zarr V2, or root `zarr.json` for Zarr V3)
- **CACHE_TILE_PRESENCE=[YES/NO]:** Defaults to `NO`. Whether to establish an initial listing of present tiles. This cached listing will be stored in a .gmac file next to the .zarray / .array.json.gmac file if they can be written. Otherwise the *GDAL_PAM_PROXY_DIR* config option should be set to an existing directory where those cached files will be stored. Once the cached listing has been established, the open option no longer needs to be specified. Note: the runtime of this option can be in minutes or more for large datasets stored on remote file systems. And for network file systems, this will rarely work for /vsicurl/ itself, but more cloud-based file systems (such as /vsis3/, /vsigs/, /vsiaz/, etc) which have a dedicated directory listing operation.
- **CACHE_KERCHUNK_JSON=[YES/NO]:** (GDAL >= 3.11) Defaults to `NO`. Whether to use (and generate if needed) a local cache where Kerchunk JSON reference files are transformed as Kerchunk Parquet reference files for more efficiency
- **MULTIBAND=[YES/NO]:** (GDAL >= 3.8) Defaults to `YES`. Whether to expose > 3D arrays as GDAL multiband datasets (when using the classic 2D API)
- **DIM_X=<string> or <integer>:** (GDAL >= 3.8) Name or index of the X dimension (only used when MULTIBAND=YES and with the classic 2D API). If not specified, deduced from dimension type (when equal to "HORIZONTAL_X"), or the last dimension (i.e. fastest varying one), if no dimension type found.
- **DIM_Y=<string> or <integer>:** (GDAL >= 3.8) Name or index of the Y dimension (only used when MULTIBAND=YES and with the classic 2D API). If not specified, deduced from dimension type (when equal to "HORIZONTAL_Y"), or the before last dimension, if no dimension type found.
- **LOAD_EXTRA_DIM_METADATA_DELAY=<integer> or "unlimited":** (GDAL >= 3.8) Defaults to `5`. Maximum delay in seconds allowed to set the DIM_{dimname}_VALUE band metadata items from the indexing variable of the dimensions. Default value is `5`. `unlimited` can be used to mean unlimited delay. Can also be defined globally with the GDAL_LOAD_EXTRA_DIM_METADATA_DELAY configuration` option. Only used through the classic 2D API.

6.160.14 Multi-threaded caching

Starting with GDAL 3.13, when the *GDAL_NUM_THREADS* configuration option is set and a read request spans multiple chunks, the driver automatically decodes chunks in parallel, similar to the GeoTIFF driver behavior (since GDAL 3.6). No application changes are needed.

The driver also implements the *GDALMDArray::AdviseRead()* method for explicit multi-threaded pre-fetching of tiles that intersect the area of interest specified. A sufficient cache size must be specified. The call is blocking.

The options that can be passed to the methods are:

- **CACHE_SIZE=value_in_byte**: Maximum RAM to use, expressed in number of bytes. If not specified, half of the remaining GDAL block cache size will be used. Note: the caching mechanism of Zarr array will not update this remaining block cache size.
- **NUM_THREADS=integer or ALL_CPUS**: Number of threads to use in parallel. If not specified, the *GDAL_NUM_THREADS* configuration option will be taken into account.

6.160.15 Creation options

The following options are creation options of the classic raster API, or array-level creation options for the multidimensional API (must be prefixed with `ARRAY:` using **gdalmdimtranslate**):

- **GEOREFERENCING_CONVENTION=[GDAL/SPATIAL_PROJ]:** Defaults to `GDAL`. Which convention is used to write georeferencing information: geotransformation and CRS. The `GDAL` convention uses a `_CRS` attribute described above. The `SPATIAL_PROJ` convention, added both in read and write support in GDAL 3.13, uses the Zarr spatial and geo-proj conventions.
- **COMPRESS=[NONE/BLOSC/ZLIB/GZIP/LZMA/ZSTD/LZ4]:** Defaults to `NONE`. Compression method. For FORMAT=ZARR_V3, only `NONE`, `BLOSC`, `GZIP` and `ZSTD` are supported.
- **FILTER=[NONE/DELTA]:** Defaults to `NONE`. Filter method. Only support for FORMAT=ZARR_V2.
- **BLOCKSIZE=<string>:** Comma separated list of chunk size along each dimension. If not specified, the fastest varying 2 dimensions (the last ones) used a block size of 256 samples, and the other ones of 1.
- **SHARD_CHUNK_SHAPE=<string>:** (GDAL >= 3.13) Comma-separated inner chunk dimensions for Zarr V3 sharded storage. When set, *BLOCKSIZE* defines the shard dimensions and this option defines the inner chunk dimensions within each shard. Each value must evenly divide the corresponding *BLOCKSIZE* dimension. For example, `BLOCKSIZE=256,256` with `SHARD_CHUNK_SHAPE=64,64` creates shards of 4x4=16 inner chunks.
- **CHUNK_MEMORY_LAYOUT=[C/F]:** Defaults to `C`. Whether to use C (row-major) order or F (column-major) order in encoded chunks. Only useful when using compression. Changing to F may improve depending on array content.
- **STRING_FORMAT=[ASCII/UNICODE]:** Defaults to `ASCII`. Whether to use the numpy type for ASCII-only strings or Unicode strings. Unicode strings take 4 byte per character.
- **DIM_SEPARATOR=<string>:** Dimension separator in chunk filenames. Default to decimal point for ZarrV2 and slash for ZarrV3.
- **BLOSC_CNAME=[bloclz/lz4/lz4hc/snappy/zlib/zstd]:** Defaults to `lz4`. Blosc compressor name. Only used when *COMPRESS=BLOSC*.
- **BLOSC_CLEVEL=1-9:** Defaults to `5`. Blosc compression level. Only used when *COMPRESS=BLOSC*.
- **BLOSC_SHUFFLE=[NONE/BYTE/BIT]:** Defaults to `BYTE`. Type of shuffle algorithm. Only used when *COMPRESS=BLOSC*.
- **BLOSC_BLOCKSIZE=<integer>:** Defaults to `0`. Blosc block size. Only used when *COMPRESS=BLOSC*.
- **BLOSC_NUM_THREADS=[<integer>/ALL_CPUS]:** Defaults to `1`. Number of worker threads for compression. Only used when *COMPRESS=BLOSC*.
- **ZLIB_LEVEL=1-9:** Defaults to `6`. ZLib compression level. Only used when *COMPRESS=ZLIB*.
- **GZIP_LEVEL=1-9:** Defaults to `6`. GZip compression level. Only used when *COMPRESS=GZIP*.
- **LZMA_PRESET=0-9:** Defaults to `6`. LZMA compression level. Only used when *COMPRESS=LZMA*.
- **LZMA_DELTA=<integer>:** Defaults to `1`. Delta distance in byte. Only used when *COMPRESS=LZMA*.
- **ZSTD_LEVEL=1-22:** Defaults to `13`. ZSTD compression level. Only used when *COMPRESS=ZSTD*.

- **LZ4_ACCELERATION=<integer> [1-]:** Defaults to 1. LZ4 acceleration factor. The higher, the less compressed. Only used when *COMPRESS=LZ4*. Defaults to 1 (the fastest).
- **DELTA_DTYPE=<string>:** Data type following NumPy array protocol type string (typestr) format (https://numpy.org/doc/stable/reference/arrays.interface.html#arrays-interface). Only `u1`, `i1`, `u2`, `i2`, `u4`, `i4`, `u8`, `i8`, `f4`, `f8`, potentially prefixed with the endianness flag (< for little endian, > for big endian) are supported. Only used when *FILTER=DELTA*. Defaults to the native data type.

The following options are creation options of the classic raster API, or dataset-level creation options for the multidimensional API :

- **FORMAT=[ZARR_V2/ZARR_V3]:** Defaults to `ZARR_V2`.
- **CREATE_CONSOLIDATED_METADATA=[YES/NO]:** Defaults to `YES`. Whether to create consolidated metadata (into `.zmetadata` for Zarr V2, or root `zarr.json` for Zarr V3)
- **CONVERT_TO_KERCHUNK_PARQUET_REFERENCE=[YES/NO]:** (GDAL >= 3.11) Defaults to `NO`. Whether to convert a Kerchunk JSON reference store into a Kerchunk Parquet reference store.

The following options are creation options of the classic raster API only:

- **ARRAY_NAME=<string>:** Array name. If not specified, deduced from the filename.
- **APPEND_SUBDATASET=[YES/NO]:** Defaults to `NO`. Whether to append the new dataset to an existing Zarr hierarchy.
- **SINGLE_ARRAY=[YES/NO]:** (GDAL >= 3.8) Defaults to `YES`. Whether to write a multi-band dataset as a 3D Zarr array. If false, one 2D Zarr array per band will be written.
- **INTERLEAVE=[BAND/PIXEL]:** (GDAL >= 3.8) Defaults to `BAND`. When writing a multi-band dataset as a 3D Zarr array, whether the band dimension should be the first one/slowest varying one (BAND), or the last one/fastest varying one (PIXEL) The default value is BAND in Create() mode. In CreateCopy() mode, the default value is the value of the INTERLEAVE metadata item of the IMAGE_STRUCTURE metadata domain of the source dataset, if set. See *Multiband pixel organization (INTERLEAVE metadata item)* for more details.

6.160.16 Add a georeferencing convention to an existing ZARR dataset

See *gdal driver zarr add-georeferencing-convention*.

6.160.17 Examples

Get information on the dataset using the multidimensional tools:

```
gdalmdiminfo my.zarr
```

Get information on the dataset using the multidimensional tools when there is no directory listing available or reliable:

```
gdalmdiminfo /vsicurl/https://example.com/my.zarr/.zmetadata
```

Convert a netCDF file to ZARR using the multidimensional tools:

```
gdalmdimtranslate in.nc out.zarr -co ARRAY:COMPRESS=GZIP
```

Convert a 2D slice (the one at index 0 of the non-2D dimension) of a 3D array to GeoTIFF:

```
gdal_translate 'ZARR:"my.zarr":/group/myarray:0' out.tif
```

Note

The single quoting around the connection string is specific to the Bash shell to make sure that the double quoting is preserved.

6.160.18 See Also:

- Zarr format and its Python implementation
- (In progress) Zarr V3 specification

6.161 ZMap -- ZMap Plus Grid

Driver short name

ZMAP

Driver built-in by default

This driver is built-in by default

Supported for read access and creation. This format is an ASCII interchange format for gridded data in an ASCII line format for transport and storage. It is commonly used in applications in the Oil and Gas Exploration field.

By default, files are interpreted and written according to the PIXEL_IS_AREA convention. If you define the ZMAP_PIXEL_IS_POINT configuration option to TRUE, the PIXEL_IS_POINT convention will be followed to interpret/write the file (the georeferenced values in the header of the file will then be considered as the coordinate of the center of the pixels). Note that in that case, GDAL will report the extent with its usual PIXEL_IS_AREA convention (the coordinates of the topleft corner as reported by GDAL will be a half-pixel at the top and left of the values that appear in the file).

Informal specification given in this GDAL-dev mailing list thread

NOTE: Implemented as zmapdataset.cpp.

6.161.1 Driver capabilities

Supports CreateCopy()

This driver supports the *`GDALDriver::CreateCopy()`* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

CHAPTER

SEVEN

VECTOR DRIVERS

Short Name	Long Name	Creation	Georeferencing	Build Req
ADBC	Arrow Database Connectivity	No	No	adbc-drive
AIVector	Artificial intelligence powered vector driver	No	No	Built-in by
AmigoCloud	AmigoCloud	**Yes**	**Yes**	libcurl
Arrow	(Geo)Arrow IPC File Format / Stream	**Yes**	**Yes**	Apache Ar
AVCBIN	Arc/Info Binary Coverage	No	**Yes**	Built-in by
AVCE00	Arc/Info E00 (ASCII) Coverage	No	**Yes**	Built-in by
CAD	AutoCAD DWG	No	**Yes**	(internal li
CARTO	Carto	**Yes**	**Yes**	libcurl
CSV	Comma Separated Value (.csv)	**Yes**	**Yes**	Built-in by
CSW	OGC CSW (Catalog Service for the Web)	No	**Yes**	libcurl
DGN	Microstation DGN	**Yes**	**Yes**	Built-in by
DGNv8	Microstation DGN v8	**Yes**	**Yes**	Open Desi
DWG	AutoCAD DWG	No	No	Open Desi
DXF	AutoCAD DXF	**Yes**	No	Built-in by
EDIGEO	EDIGEO	No	**Yes**	Built-in by
EEDA	Google Earth Engine Data API	No	**Yes**	libcurl
Elasticsearch	Elasticsearch: Geographically Encoded Objects for Elasticsearch	**Yes**	**Yes**	libcurl
ESRI Shapefile	ESRI Shapefile / DBF	**Yes**	**Yes**	Built-in by
ESRIJSON	ESRIJSON / FeatureService driver	No	**Yes**	Built-in by
FileGDB	ESRI File Geodatabase (FileGDB)	**Yes**	**Yes**	FileGDB A
FlatGeobuf	FlatGeobuf	**Yes**	**Yes**	Built-in by
GDALG	GDALG: GDAL Streamed Algorithm	No	**Yes**	Built-in by
GeoJSON	GeoJSON	**Yes**	**Yes**	Built-in by
GeoJSONSeq	GeoJSONSeq: sequence of GeoJSON features	**Yes**	**Yes**	Built-in by
GeoRSS	GeoRSS : Geographically Encoded Objects for RSS feeds	**Yes**	**Yes**	(read supp
GML	Geography Markup Language	**Yes**	**Yes**	(read supp
GMLAS	Geography Markup Language (GML) driven by application schemas	No	**Yes**	Xerces
GMT	GMT ASCII Vectors (.gmt)	**Yes**	**Yes**	Built-in by
GPKG	GeoPackage vector	**Yes**	**Yes**	libsqlite3
GPSBabel	GPSBabel	**Yes**	**Yes**	(read supp
GPX	GPS Exchange Format	**Yes**	**Yes**	(read supp
GRASS	GRASS Vector Format	No	No	gdal-grass-
GTFS	General Transit Feed Specification	No	No	Built-in by
HANA	SAP HANA	**Yes**	**Yes**	odbc-cpp-v
IDB	IDB	**Yes**	**Yes**	Informix D
IDRISI	Idrisi Vector (.VCT)	No	**Yes**	Built-in by
INTERLIS 1	"INTERLIS 1" and "INTERLIS 2" drivers	No	**Yes**	Xerces
JML	JML: OpenJUMP JML format	**Yes**	**Yes**	(read supp

Table 1 – continued from previous page

Short Name	Long Name	Creation	Georeferencing	Build Req
JSONFG	OGC Features and Geometries JSON	**Yes**	**Yes**	Built-in by
KML	Keyhole Markup Language	**Yes**	**Yes**	(read supp
LIBKML	LIBKML Driver (.kml .kmz)	**Yes**	**Yes**	libkml
LVBAG	Dutch Kadaster LV BAG 2.0 Extract	No	No	libexpat
MapInfo File	MapInfo TAB and MIF/MID	**Yes**	**Yes**	Built-in by
MapML	MapML	**Yes**	**Yes**	Built-in by
MEM	In Memory datasets	**Yes**	**Yes**	Built-in by
Memory	Memory (deprecated)	No	No	Built-in by
MiraMonVector	MiraMon Vector	**Yes**	**Yes**	Built-in by
MongoDBv3	MongoDBv3	No	**Yes**	Mongo CX
MSSQLSpatial	Microsoft SQL Server Spatial Database	**Yes**	**Yes**	ODBC libr
MVT	MVT: Mapbox Vector Tiles	**Yes**	**Yes**	(requires S
MySQL	MySQL	**Yes**	**Yes**	MySQL lib
NAS	ALKIS	No	**Yes**	Xerces
netCDF	NetCDF: Network Common Data Form - Vector	**Yes**	**Yes**	libnetcdf
NGW	NextGIS Web	No	**Yes**	libcurl
OAPIF	OGC API - Features	No	**Yes**	libcurl
OCI	Oracle Spatial	**Yes**	**Yes**	OCI library
ODBC	ODBC RDBMS	No	**Yes**	ODBC libr
ODS	Open Document Spreadsheet	**Yes**	No	libexpat
OpenFileGDB	ESRI File Geodatabase vector (OpenFileGDB)	**Yes**	**Yes**	Built-in by
OSM	OpenStreetMap XML and PBF	No	**Yes**	libsqlite3 (
Parquet	(Geo)Parquet	**Yes**	**Yes**	Parquet co
PDF	Geospatial PDF	**Yes**	**Yes**	none for w
PDS	Planetary Data Systems TABLE	No	No	Built-in by
PGDump	PostgreSQL SQL Dump	**Yes**	**Yes**	Built-in by
PGeo	ESRI Personal GeoDatabase	No	**Yes**	ODBC libr
PLScenes	PLScenes (Planet Labs Scenes/Catalog API)	No	No	libcurl
PMTiles	PMTiles	**Yes**	**Yes**	Built-in by
PostgreSQL	PostgreSQL / PostGIS	**Yes**	**Yes**	PostgreSQ
S101	IHO S-101 (ENC)	No	**Yes**	Built-in by
S57	IHO S-57 (ENC)	**Yes**	**Yes**	Built-in by
Selafin	Selafin files	**Yes**	**Yes**	Built-in by
SOSI	Norwegian SOSI Standard	No	No	FYBA libr
SQLite	SQLite / Spatialite RDBMS	**Yes**	**Yes**	libsqlite3 c
SXF	SXF	No	**Yes**	Built-in by
TIGER	U.S. Census TIGER/Line	No	**Yes**	Built-in by
TileDB	TileDB vector	**Yes**	No	TileDB >=
TopoJSON	TopoJSON driver	No	**Yes**	Built-in by
UK .NTF	UK .NTF	No	**Yes**	Built-in by
VDV	VDV-451/VDV-452/INTREST Data Format	**Yes**	**Yes**	Built-in by
VFK	Czech Cadastral Exchange Data Format	No	**Yes**	libsqlite3
VRT	OGR Virtual Format	No	**Yes**	Built-in by
WAsP	WAsP .map format	**Yes**	**Yes**	Built-in by
WFS	OGC WFS service	No	**Yes**	libcurl
XLS	MS Excel format	No	No	libfreexl
XLSX	MS Office Open XML spreadsheet	**Yes**	No	libexpat
XODR	OpenDRIVE Road Description Format	No	**Yes**	libOpenDR

Note

The following drivers have been removed in GDAL 3.5: AeronavFAA, BNA, HTF, OpenAir, REC, SEGUKOOA, SEGY, SUA, XPlane

The following drivers have been removed in GDAL 3.11: Geoconcept Export, OGDI (VPF/VMAP support), SDTS, SVG

7.1 ADBC -- Arrow Database Connectivity

Added in version 3.11.

Driver short name

ADBC

Build dependencies

adbc-driver-manager

ADBC is a set of APIs and libraries for Arrow-native access to database.

This driver has 2 modes:

- either it has been built against the `adbc-driver-manager` library. In that case, it can directly be used to connect to available ADBC drivers, and expose content as classic OGR features, or as a ArrowArrayStream. In that mode the driver metadata exposes the `HAS_ADBC_DRIVER_MANAGER` metadata item.
- or it has not, in which case applications embedding GDAL must use *`GDALSetAdbcLoadDriverOverride()`* as detailed in a below paragraph. Note that use of that function can also be done even if the driver has been built against the `adbc-driver-manager` library.

Consult the installation instruction for the various ADBC drivers. At time of writing, there are drivers for SQLite3, PostgreSQL, Snowflake, BigQuery, DuckDB, Flight SQL, etc.

The driver is read-only.

There is spatial support when the underlying ADBC driver is DuckDB, for native spatial DuckDB databases and GeoParquet datasets, and when the spatial extension is installed.

7.1.1 Google BigQuery support

Consult the *ADBC BigQuery -- Google BigQuery through Arrow Database Connectivity* page.

7.1.2 Connection string

Several connection strings are supported:

- `ADBC:{some_uri}`, together with the `ADBC_DRIVER` open option.
- a SQLite3 database filename, if the `adbc_driver_sqlite` is available.
- a DuckDB database filename, if the `libduckdb.so`, `libduckdb.dylib` or `duckdb.dll` is available (and it is in a system location, or can be located through LD_LIBRARY_PATH on Linux, DYLD_LIBRARY_PATH on MacOSX or PATH on Windows).

- a Parquet database filename, if the `libduckdb.so`, `libduckdb.dylib` or `duckdb.dll` is available (and it is in a system location, or can be located through LD_LIBRARY_PATH on Linux, DYLD_LIBRARY_PATH on MacOSX or PATH on Windows).
- a PostgreSQL URI starting with `postgresql://`, if the `adbc_driver_postgresql` is available.

Note: if present, the *SQLite / Spatialite RDBMS*, *GPKG -- GeoPackage vector* or *(Geo)Parquet* drivers are registered before the ADBC driver, so they will be used in priority when available. The `ADBC:` prefix or the `-if ADBC` switch of **ogrinfo** or **ogr2ogr** can be used to use the ADBC driver instead.

7.1.3 Dataset open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are supported:

- **ADBC_DRIVER=<string>:** ADBC driver name. Examples: `adbc_driver_sqlite`, `adbc_driver_postgresql`, `adbc_driver_bigquery`, `adbc_driver_snowflake` or a path to the DuckDB shared library.
- **SQL=<string>:** A SQL-like statement recognized by the driver, used to create a result layer from the dataset.
- **ADBC_OPTION_xxx=<string>:** Custom ADBC option to pass to AdbcDatabaseSetOption(). Options are driver specific. For example `ADBC_OPTION_uri=some_value` to pass the `uri` option.
- **PRELUDE_STATEMENTS=<string>:** SQL-like statement recognized by the driver that must be executed before discovering layers. Can be repeated multiple times. For example `PRELUDE_STATEMENTS=INSTALL spatial` and `PRELUDE_STATEMENTS=LOAD spatial` to load DuckDB spatial extension.

7.1.4 "table_list" special layer

For PostgreSQL, SQLite3, DuckDB and Parquet datasets, the driver automatically instantiates OGR layers from available tables. For other databases, the user must explicit provide a SQL open option or issue a *GDALDataset::ExecuteSQL()* request. To facilitate that process, a special OGR `table_list` layer can be queried through *GDALDataset::GetLayerByName()* (or as the layer name with **ogrinfo**). It returns for each table a OGR feature with the following fields (some potentially unset or with an empty string): `catalog_name`, `schema_name`, `table_name`, `table_type`.

7.1.5 Custom driver entry point

A custom driver entry point can be specified by applications by calling *GDALSetAdbcLoadDriverOverride()* (defined in header `gdal_adbc.h`) before using the driver. The specified init function will be used by the GDAL ADBC driver as a way of locating and loading the ADBC driver if GDAL was not built with ADBC Driver Manager support or if an embedding application has an updated or augmented collection of drivers available.

7.1.6 Filtering

Attribute filters are passed to the underlying ADBC engine.

Spatial filters are passed to DuckDB when it is the underlying ADBC engine and for DuckDB spatial databases and GeoParquet datasets. GeoParquet bounding box column and/or DuckDB native RTree spatial indices are used when available.

7.1.7 Examples

7.1.7.1 Example 1: Convert a Parquet file to GeoPackage

Assuming `libduckdb.so`, `libduckdb.dylib` or `duckdb.dll` is available (and it is in a system location, or can be located through LD_LIBRARY_PATH on Linux, DYLD_LIBRARY_PATH on MacOSX or PATH on Windows).

```
ogr2ogr out.gpkg in.parquet
```

7.1.7.2 Example 2: Convert a DuckDB database to GeoPackage

Assuming `libduckdb.so`, `libduckdb.dylib` or `duckdb.dll` is available (and it is in a system location, or can be located through LD_LIBRARY_PATH on Linux, DYLD_LIBRARY_PATH on MacOSX or PATH on Windows).

```
ogr2ogr out.gpkg in.duckdb
```

7.1.8 See Also

ADBC: Arrow Database Connectivity

7.2 ADBC BigQuery -- Google BigQuery through Arrow Database Connectivity

Added in version 3.12.

Driver short name

ADBC

Build dependencies

adbc-driver-manager

See *ADBC -- Arrow Database Connectivity* for generalities on the ADBC driver.

The ADBC driver supports read and write operations to BigQuery tables,

7.2.1 Runtime requirements

adbc_driver_bigquery must be available in a path from which shared libraries can be loaded (for example by setting the `LD_LIBRARY_PATH` environment variable on Linux, `DYLD_LIBRARY_PATH` on MacOS X or `PATH` on Windows). Note that compilation of that driver requires a Go compiler.

7.2.2 Dataset connection string

The `ADBC:` connection string must be provided, as well with the *BIGQUERY_PROJECT_ID* open option and at least one of *BIGQUERY_JSON_CREDENTIAL_STRING* or *BIGQUERY_JSON_CREDENTIAL_FILE*.

7.2.3 Dataset open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are supported:

- **ADBC_DRIVER=<string>:** ADBC driver name. Must be set to `adbc_driver_bigquery` for BigQuery support. It is not necessary to explicitly set it, if at least one of the below open option starting with `BIGQUERY\_` is set.
- **SQL=<string>:** A SQL-like statement recognized by the driver, used to create a result layer from the dataset.

- **BIGQUERY_PROJECT_ID=<string>:** Required. Name of the Google project ID This can also be set as a configuration option.
- **BIGQUERY_DATASET_ID=<string>:** Optional if providing the SQL open option. Required otherwise. Name of the BigQuery dataset ID This can also be set as a configuration option.
- **BIGQUERY_JSON_CREDENTIAL_STRING=<string>:** JSON string containing Google credentials. This is typically the content of the `$HOME/.config/gcloud/application_default_credentials.json` file generated with `gcloud auth application-default login`. Mutually exclusive with BIGQUERY_JSON_CREDENTIAL_FILE. One of them required. This can also be set as a configuration option.
- **BIGQUERY_JSON_CREDENTIAL_FILE=<string>:** Filename containing Google credentials. It may be created with `gcloud auth application-default login`, which will generate a `$HOME/.config/gcloud/application_default_credentials.json` file. Mutually exclusive with BIGQUERY_JSON_CREDENTIAL_STRING. One of them required. This can also be set as a configuration option.
- **ADBC_OPTION_xxx=<string>:** Custom ADBC option to pass to AdbcDatabaseSetOption(). Options are driver specific. For example `ADBC_OPTION_uri=some_value` to pass the `uri` option.
- **PRELUDE_STATEMENTS=<string>:** SQL-like statement recognized by the driver that must be executed before discovering layers. Can be repeated multiple times. For example `PRELUDE_STATEMENTS=INSTALL spatial` and `PRELUDE_STATEMENTS=LOAD spatial` to load DuckDB spatial extension.

7.2.4 Write support

The following operations are supported (provided appropriate permissions are set on server side):

- creating a new table
- deleting a table
- adding a new field to a table
- adding a new feature to a table
- updating an existing feature in a table
- deleting an existing feature from a table.

Note that feature insertion is done one feature at a time, so it is only appropriate for creation of a few features. Use BigQuery bulk import capabilities when importing a substantial number of features.

7.2.5 Layer creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALDatasetCreateLayer()`* (C) or *`Dataset.CreateLayer`* (Python). The following layer creation options are available:

- **FID=value:** Defaults to `ogc_fid`. Column name to use for the OGR FID (integer primary key in the table). Can be set to empty to ask for not creating a primary key (in which case operations like updating or deleting existing features will not be available through *`OGRLayer::SetFeature()`* and *`OGRLayer::DeleteFeature()`*)

7.2.6 Examples

7.2.6.1 Example 1: Listing available tables from a BigQuery dataset

```
gdal vector info ADBC: \
    --oo BIGQUERY_PROJECT_ID=my_project_id \
    --oo BIGQUERY_DATASET_ID=my_dataset \
    --oo BIGQUERY_JSON_CREDENTIAL_FILE=$HOME/.config/gcloud/application_default_
↪credentials.json
```

7.2.6.2 Example 2: Displaying the results of a SQL statement

```
gdal vector info ADBC: --features \
    --oo BIGQUERY_PROJECT_ID=my_project_id \
    --oo BIGQUERY_JSON_CREDENTIAL_FILE=$HOME/.config/gcloud/application_default_
↪credentials.json
    --oo "SQL=SELECT * FROM my_dataset.my_table ORDER BY area DESC LIMIT 10"
```

7.2.6.3 Example 3: Converting a subset of a BigQuery table to a GeoPackage file

```
gdal vector sql --input=ADBC: --output=out.gpkg \
    "--sql=SELECT * FROM my_dataset.my_table WHERE country = 'France'" \
    --oo BIGQUERY_PROJECT_ID=my_project_id \
    --oo BIGQUERY_JSON_CREDENTIAL_FILE=$HOME/.config/gcloud/application_default_
↪credentials.json
```

7.2.6.4 Example 4: Converting a GeoPackage file to a BigQuery table (provided the input table(s) use geographic coordinates):

```
gdal vector convert --update --input=my.gpkg --output=ADBC: \
    --output-oo BIGQUERY_PROJECT_ID=my_project_id \
    --output-oo BIGQUERY_DATASET_ID=my_dataset \
    --output-oo BIGQUERY_JSON_CREDENTIAL_FILE=$HOME/.config/gcloud/application_default_
↪credentials.json
```

7.2.6.5 Example 5: Converting a shapefile file to a BigQuery table, doing prior reprojection to WGS 84 (EPSG:4326), and renaming the layer

```
gdal vector pipeline read my.gpkg ! \
    reproject --dst-crs=EPSG:4326 ! \
    write ADBC: \
        --update \
        --output-oo BIGQUERY_PROJECT_ID=my_project_id \
        --output-oo BIGQUERY_DATASET_ID=my_dataset \
        --output-oo BIGQUERY_JSON_CREDENTIAL_FILE=$HOME/.config/gcloud/application_
↪default_credentials.json \
        --output-layer newlayer
```

7.2.7 See Also

- adbc_driver_bigquery

7.3 Artificial intelligence powered vector driver

Added in version 3.11.

Driver short name

AIVector

Driver built-in by default

This driver is built-in by default

This driver builds on many years of self-funded investments from the GDAL team on AI technologies to bring you the ultimate driver that can read any vector format. After that one, no need for any new vector driver!

The open syntax is `AIVector:{filename}`, or directly specify the filename and force the use of the AIVector driver with the `-if` flag of ogrinfo or ogr2ogr. No options at all. Just enjoy the true power of AI.

Note

We are open to external investors to develop the write side of the driver.

7.3.1 Examples

```
ogrinfo -if AIVector undocumented_proprietary_format.bin -al
```

Note

The above works even if you make a typo in the filename. The driver will automatically figure out the filename you meant.

7.4 AmigoCloud

Driver short name

AmigoCloud

Build dependencies

libcurl

This driver can connect to the AmigoCloud API services. GDAL/OGR must be built with Curl support in order for the AmigoCloud driver to be compiled.

The driver supports read and write operations.

7.4.1 Driver capabilities

Supports Create()

This driver supports the *`GDALDriver::Create()`* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.4.2 Dataset name syntax

The minimal syntax to open a AmigoCloud datasource is:

```
AmigoCloud:[project_id]
```

Additional optional parameters can be specified after the ':' sign. Currently the following one is supported :

- **datasets=dataset_id1[,dataset_id2, ..]**: A list of AmigoCloud dataset IDs. This is necessary when you need to access a particular AmigoCloud dataset.

If several parameters are specified, they must be separated by a space.

If no dataset_id is provided, the driver will print list of available datasets for given project.

For example: **"AmigoCloud:1234 datasets"**

```
List of available datasets for project id: 1234
| id        | name
|-----------|-------------------
| 5551      | points
| 5552      | lines
```

7.4.3 Authentication

All the access permissions are defined by AmigoCloud backend.

Authenticated access is obtained by specifying the API key given in the AmigoCloud dashboard web interface. It is specified with the *AMIGOCLOUD_API_KEY* configuration option.

7.4.4 Geometry

The OGR driver will report as many geometry fields as available in the layer, following RFC 41.

7.4.5 Filtering

The driver will forward any spatial filter set with *OGRLayer::SetSpatialFilter()* to the server. It also makes the same for attribute filters set with *OGRLayer::SetAttributeFilter()*.

7.4.6 Write support

Dataset creation and deletion is possible.

Write support is only enabled when the datasource is opened in update mode.

The mapping between the operations of the AmigoCloud service and the OGR concepts is the following :

- *OGRLayer::CreateFeature()* <==> INSERT operation

- *OGRLayer::SetFeature()* <==> UPDATE operation
- *OGRLayer::DeleteFeature()* <==> DELETE operation
- *GDALDataset::CreateLayer()* <==> CREATE TABLE operation
- *GDALDataset::DeleteLayer()* <==> DROP TABLE operation

When inserting a new feature with CreateFeature(), and if the command is successful, OGR will fetch the returned amigo_id (GUID) and use hash value of it as the OGR FID.

The above operations are by default issued to the server synchronously with the OGR API call. This however can cause performance penalties when issuing a lot of commands due to many client/server exchanges.

7.4.7 Layer creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALDatasetCreateLayer()* (C) or *Dataset.CreateLayer* (Python). The following layer creation options are available:

- **OVERWRITE=[YES/NO]:** Defaults to NO. Whether to overwrite an existing table with the layer name to be created.
- **GEOMETRY_NULLABLE=[YES/NO]:** Defaults to YES. Whether the values of the geometry column can be NULL.

7.4.8 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). The following configuration options are available:

- **AMIGOCLOUD_API_URL=value:** Defaults to `https://www.amigocloud.com/api/v1`. Can be used to point to another server.
- **AMIGOCLOUD_API_KEY=value:** See usage examples.

7.4.9 Examples

7.4.9.1 Example 1: Different ways to provide AmigoCloud API token

```
ogrinfo --config AMIGOCLOUD_API_KEY abcdefghijklmnopqrstuvw -al "AmigoCloud:1234␣
↪datasets=987"
ogrinfo -oo AMIGOCLOUD_API_KEY=abcdefghijklmnopqrstuvw -al "AmigoCloud:1234 datasets=987"
env AMIGOCLOUD_API_KEY=abcdefghijklmnopqrstuvw ogrinfo -al "AmigoCloud:1234 datasets=987"
```

```
export AMIGOCLOUD_API_KEY=abcdefghijklmnopqrstuvw
ogrinfo -al "AmigoCloud:1234 datasets=987"
```

7.4.9.2 Example 2: Show list of datasets

```
$ ogrinfo -ro "AmigoCloud:1234 datasets"
List of available datasets for project id: 1234
| id         | name
|-----------|-------------------
| 5551       | points
| 5552       | lines
```

7.4.9.3 Example 3: Accessing data from a list of datasets

```
ogrinfo -ro "AmigoCloud:1234 datasets=1234,1235"
```

7.4.9.4 Example 4: Creating and populating a table from a shapefile

```
ogr2ogr -f AmigoCloud "AmigoCloud:1234" myshapefile.shp
```

7.4.9.5 Example 5: Append the data to an existing table (dataset id: 12345) from a shapefile

```
ogr2ogr -f AmigoCloud "AmigoCloud:1234 datasets=12345" myshapefile.shp
```

or

```
ogr2ogr -append -f AmigoCloud "AmigoCloud:1234 datasets=12345" myshapefile.shp
```

7.4.9.6 Example 6: Overwriting the data of an existing table (dataset id: 12345) with data from a shapefile

```
ogr2ogr -append -doo OVERWRITE=YES -f AmigoCloud "AmigoCloud:1234 datasets=12345"␣
↪myshapefile.shp
```

7.4.9.7 Example 7: Delete existing dataset (dataset id: 12345) and create a new one with data from a shapefile

```
ogr2ogr -overwrite -f AmigoCloud "AmigoCloud:1234 datasets=12345" myshapefile.shp
```

7.4.9.8 Example 8: Overwriting the data of an existing table (dataset id: 12345) with data from a shapefile.

Filter the only the records with values of the field "visited_on" after 2017-08-20

```
ogr2ogr -append -doo OVERWRITE=YES -f AmigoCloud "AmigoCloud:1234 datasets=12345" -where
↪"visited_on > '2017-08-20'" myshapefile.shp
```

7.4.10 See Also

- AmigoCloud API Token management
- AmigoCloud API Browser

7.5 (Geo)Arrow IPC File Format / Stream

Added in version 3.5.

Driver short name

Arrow

Build dependencies

Apache Arrow C++ library

The Arrow IPC File Format (Feather) is a portable file format for storing Arrow tables or data frames (from languages like Python or R) that utilizes the Arrow IPC format internally.

The driver supports the 2 variants of the format:

- File or Random Access format, also known as Feather: for serializing a fixed number of record batches. Random access is required to read such files, but they can be generated using a streaming-only capable file. The recommended extension for such file is `.arrow`
- Streaming IPC format: for sending an arbitrary length sequence of record batches. The format must generally be processed from start to end, and does not require random access. That format is not generally materialized as a file. If it is, the recommended extension is `.arrows` (with a trailing s). But the driver can support regular files as well as the /vsistdin/ and /vsistdout/ streaming files. On opening, it might difficult for the driver to detect that the content is specifically a Arrow IPC stream, especially if the extension is not `.arrows`, and the metadata section is large. Prefixing the filename with `ARROW_IPC_STREAM:` (e.g "ARROW_IPC_STREAM:/vsistdin/") will cause the driver to unconditionally open the file as a streaming IPC format. Alternatively, starting with GDAL 3.10, specifying the `-if ARROW` option to command line utilities accepting it, or `ARROW` as the only value of the `papszAllowedDrivers` of *`GDALOpenEx()`*, also forces the driver to recognize the passed filename.

This driver also supports geometry columns using the GeoArrow specification.

Note

The driver should be considered experimental as the GeoArrow specification is not finalized yet.

7.5.1 Driver capabilities

Supports Create()

This driver supports the *`GDALDriver::Create()`* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.5.2 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are supported:

- **LISTS_AS_STRING_JSON=[YES/NO]:** (GDAL >= 3.12.1) Defaults to `NO`. Whether lists of strings/integers/reals should be reported as String(JSON) fields rather than String/Integer[64]/RealList.

Useful when null values in such lists must be exactly mapped as such, instead of being omitted (for lists of strings), or set to 0 (for list of boolean, integer or real).

7.5.3 Creation issues

The driver supports creating only a single layer in a dataset.

7.5.4 Layer creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALDatasetCreateLayer()* (C) or *Dataset.CreateLayer* (Python). The following layer creation options are supported:

- **COMPRESSION=[NONE/ZSTD/LZ4]:** Compression method. Available values depend on how the Arrow library was compiled. Defaults to LZ4 when available, otherwise NONE.
- **FORMAT=[FILE/STREAM]:** Variant of the file format. See introduction paragraph for the difference between both. Defaults to FILE, unless the filename is "/vsistdout/" or its extension is ".arrows", in which case STREAM is used.
- **GEOMETRY_ENCODING=[GEOARROW/WKB/WKT/GEOARROW_INTERLEAVED]:** Defaults to `GEOARROW`. Geometry encoding. As of GDAL 3.9, GEOARROW uses the GeoArrow "struct" based encodings (where points are modeled as a struct field with a x and y subfield, lines are modeled as a list of such points, etc.). The GEOARROW_INTERLEAVED option has been renamed in GDAL 3.9 from what was named GEOARROW in previous versions, and uses an encoding where points uses a FixedSizedList of (x,y), lines a variable-size list of such FixedSizedList of points, etc.
- **BATCH_SIZE=<integer>:** Defaults to `65536`. Maximum number of rows per record batch.
- **GEOMETRY_NAME=value:** Defaults to `geometry`. Name of geometry column.
- **FID=value:** Name of the FID (Feature Identifier) column to create. If none is specified, no FID column is created. Note that if using ogr2ogr with the Arrow driver as the target driver and a source layer that has a named FID column, this FID column name will be automatically used to set the FID layer creation option of the Arrow driver (unless `-lco FID=` is used to set an empty name)
- **TIMESTAMP_WITH_OFFSET=[AUTO/YES/NO]:** (GDAL >= 3.13) Defaults to `AUTO`. Whether OGR datetime fields should be written as Arrow timestamp with offset fields, following the Timestamp With Offset extension specification. Such fields store both the datetime as a timestamp expressed in the UTC timezone and the offset to UTC of the timezone in which the datetime is defined. In AUTO mode, they are used as soon as a DateTime field reports a mixed time zone flag (i.e. *OGRFieldDefn::GetTZFlag()* returns `OGR_TZFLAG_MIXED_TZ`). As few drivers are able to automatically set this flag, it may be useful to override the flag by setting this option to YES. Setting it to NO forces the use of a DateTime field with the UTC timezone.

7.5.5 Conda-forge package

The driver can be installed as a plugin for the `libgdal` conda-forge package with:

```
conda install -c conda-forge libgdal-arrow-parquet
```

7.5.6 Standalone plugin compilation

Added in version 3.10.

While this driver may be built as part of a whole GDAL build, either in libgdal itself, or as a plugin, it is also possible to only build this driver as a plugin, against an already built libgdal.

The version of the GDAL sources used to build the driver must match the version of the libgdal it is built against.

For example, from a "build_arrow" directory under the root of the GDAL source tree:

```
cmake -S ../ogr/ogrsf_frmts/parquet -DCMAKE_PREFIX_PATH=/path/to/GDAL_installation_
↪prefix -DArrow_DIR=/path/to/lib/cmake/Arrow
cmake --build .
```

Note that such a plugin, when used against a libgdal not aware of it, will be systematically loaded at GDAL driver initialization time, and will not benefit from *deferred plugin loading capabilities <rfc-96>*. For that, libgdal itself must be built with the CMake variable OGR_REGISTER_DRIVER_ARROW_FOR_LATER_PLUGIN=ON set.

7.5.7 Arrow VSI file system

Added in version 3.10.

Starting with GDAL 3.10 and Arrow 16.0, any GDAL Virtual File System can be used (in a read-only context) wherever the Arrow C++ library expects a URI, in particular outside of the context of the OGR Arrow driver, by:

- loading the libgdal.so/dll library (or the ogr_Arrow.so/dll plugin library if the Arrow driver is built as a library) with the arrow::fs::LoadFileSystemFactories() function (cf Defining new filesystems) Note: if the Arrow driver is fully loaded, e.g. by querying GetGDALDriverManager()->GetDriverByName("ARROW")->GetMetadata(), the Arrow VSI file system will be also registered.
- Prefixing any GDAL file name with the `gdalvsi://` URI scheme prefix. In addition to any potential vsi prefix in the GDAL file name. So the `/vsicurl/http://example.com` GDAL file name becomes the `gdalvsi://` `/vsicurl/http://example.com` Arrow URI.

7.5.8 Links

- Feather File Format
- GeoArrow specification
- Related driver: *Parquet driver*

7.6 Arc/Info Binary Coverage

Driver short name

AVCBIN

Driver built-in by default

This driver is built-in by default

Arc/Info Binary Coverages (eg. Arc/Info V7 and earlier) are supported by OGR for read access.

The label, arc, polygon, centroid, region and text sections of a coverage are all supported as layers. Attributes from INFO are appended to labels, arcs, polygons or region where appropriate. When available the projection information is read and translated. Polygon geometries are collected for polygon and region layers from the composing arcs.

Text sections are represented as point layers. Display height is preserved in the HEIGHT attribute field; however, other information about text orientation is discarded.

Info tables associated with a coverage, but not specifically named to be attached to one of the existing geometric layers is currently not accessible through OGR. Note that info tables are stored in an 'info' directory at the same level as the

coverage directory. If this is inaccessible or corrupt no info attributes will be appended to coverage layers, but the geometry should still be accessible.

If the directory contains files with names like w001001.adf then the coverage is a *grid coverage* suitable to read with GDAL, not a vector coverage supported by OGR.

The layers are named as follows:

1. A label layer (polygon labels, or free standing points) is named LAB if present.
2. A centroid layer (polygon centroids) is named CNT if present.
3. An arc (line) layer is named ARC if present.
4. A polygon layer is named "PAL" if present.
5. A text section is named according to the section subclass.
6. A region subclass is named according to the subclass name.

The Arc/Info binary coverage driver attempts to optimize spatial queries but due to the lack of a spatial index this is just accomplished by minimizing processing for features not within the spatial window.

Random (by FID) reads of arcs, and polygons is supported it may not be supported for other feature types.

7.6.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.6.2 See Also

- AVCE00 Library Page
- *AVCE00 OGR Driver (.E00)*

7.7 Arc/Info E00 (ASCII) Coverage

Driver short name

AVCE00

Driver built-in by default

This driver is built-in by default

Arc/Info E00 Coverages (eg. Arc/Info V7 and earlier) are supported by OGR for read access.

The label, arc, polygon, centroid, region and text sections of a coverage are all supported as layers. Attributes from INFO are appended to labels, arcs, polygons or region where appropriate. When available the projection information is read and translated. Polygon geometries are collected for polygon and region layers from the composing arcs.

Text sections are represented as point layers. Display height is preserved in the HEIGHT attribute field; however, other information about text orientation is discarded.

Info tables associated with a coverage, but not specifically named to be attached to one of the existing geometric layers is currently not accessible through OGR. Note that info tables are stored in an 'info' directory at the same level as the coverage directory. If this is inaccessible or corrupt no info attributes will be appended to coverage layers, but the geometry should still be accessible.

The layers are named as follows:

1. A label layer (polygon labels, or free standing points) is named LAB if present.
2. A centroid layer (polygon centroids) is named CNT if present.
3. An arc (line) layer is named ARC if present.
4. A polygon layer is named "PAL" if present.
5. A text section is named according to the section subclass.
6. A region subclass is named according to the subclass name.

Random (by FID) reads of arcs, and polygons is supported it may not be supported for other feature types. Random access to E00 files is generally slow.

7.7.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.7.2 See Also

- AVCE00 Library Page
- *AVCBin OGR Driver (Binary Coverage)*

7.8 CAD -- AutoCAD DWG

Driver short name

CAD

Build dependencies

(internal libopencad provided)

OGR DWG support is based on libopencad, so the list of supported DWG (DXF) versions can be seen in libopencad documentation. All drawing entities are separated into layers as they are in DWG file, not in 1 layer as DXF Driver does.

DWG files are considered to have no georeferencing information through OGR. Features will all have the following generic attributes:

- CADGeometry: CAD Type of the presented geometry.
- Thickness: Thickness of the object drawing units (if it is not supported by this type, it is set to 0.0).
- Color (RGB): IntegerList contains R,G,B components of the color.
- ExtendedEntity: Where available, extended entity attributes all appended to form a single text attribute.

7.8.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.8.2 Supported Elements

The following element types are supported:

- POINT: Produces a simple point geometry feature.
- LINE: Translated as a LINESTRING. Rounded polylines (those with their vertices' budge attributes set) will be tessellated. Single-vertex polylines are translated to POINT.
- CIRCLE, ARC: Translated as a CIRCULARSTRING.
- 3DFACE: Translated as POLYGON.

The driver is read-only.

7.8.3 See Also

- ODA DWG Reference
- Libopencad repository

7.9 Carto

Driver short name

CARTO

Build dependencies

libcurl

This driver can connect to the services implementing the Carto API. GDAL/OGR must be built with Curl support in order for the Carto driver to be compiled.

The driver supports read and write operations.

7.9.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

7.9.2 Dataset name syntax

The minimal syntax to open a Carto datasource is :

```
Carto:[connection_name]
```

For single-user accounts, connection name is the account name. For multi-user accounts, connection_name must be the user name, not the account name. Additional optional parameters can be specified after the ':' sign. Currently the following one is supported:

- **tables=table_name1[,table_name2]***: A list of table names. This is necessary when you need to access to public tables for example.

If several parameters are specified, they must be separated by a space.

7.9.3 Authentication

Most operations, in particular write operations, require an authenticated access. The only exception is read-only access to public tables.

Authenticated access is obtained by specifying the API key given in the management interface of the Carto service. It is specified with the CARTO_API_KEY configuration option.

7.9.4 Geometry

The OGR driver will report as many geometry fields as available in the layer (except the 'the_geom_webmercator' field), following RFC 41.

7.9.5 Filtering

The driver will forward any spatial filter set with *OGRLayer::SetSpatialFilter()* to the server. It also makes the same for attribute filters set with *OGRLayer::SetAttributeFilter()*.

7.9.6 Paging

Features are retrieved from the server by chunks of 500 by default. This number can be altered with the *CARTO_PAGE_SIZE* configuration option.

7.9.7 Write support

Table creation and deletion is possible.

Write support is only enabled when the datasource is opened in update mode.

The mapping between the operations of the Carto service and the OGR concepts is the following :

- *OGRLayer::CreateFeature()* <==> `INSERT` operation
- *OGRLayer::SetFeature()* <==> `UPDATE` operation
- *OGRLayer::DeleteFeature()* <==> `DELETE` operation
- *GDALDataset::CreateLayer()* <==> `CREATE TABLE` operation
- *GDALDataset::DeleteLayer()* <==> `DROP TABLE` operation

When inserting a new feature with *OGRLayer::CreateFeature()*, and if the command is successful, OGR will fetch the returned rowid and use it as the OGR FID.

The above operations are by default issued to the server synchronously with the OGR API call. This however can cause performance penalties when issuing a lot of commands due to many client/server exchanges.

So, on a newly created layer, the `INSERT` of *OGRLayer::CreateFeature()* operations are grouped together in chunks until they reach 15 MB (can be changed with the *CARTO_MAX_CHUNK_SIZE* configuration option, with a value in MB), at which point they are transferred to the server. By setting *CARTO_MAX_CHUNK_SIZE* to 0, immediate transfer occurs.

> **Warning**
>
> Don't use *GDALDataset::DeleteLayer()* and *GDALDataset::CreateLayer()* to overwrite a table. Instead only call *GDALDataset::CreateLayer()* with OVERWRITE=YES. This will avoid CARTO deleting maps that depend on this table

7.9.8 SQL

SQL commands provided to the *GDALDataset::ExecuteSQL()* call are executed on the server side, unless the OGRSQL dialect is specified. You can use the full power of PostgreSQL + PostGIS SQL capabilities.

7.9.9 Open options

The following open options are available: Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python).

- **BATCH_INSERT=[YES/NO]:** Defaults to `YES`. Whether to group feature insertions in a batch. Only apply in creation or update mode.
- **COPY_MODE=[YES/NO]:** Defaults to `YES`. Using COPY for insertions and reads can result in a performance improvement.

7.9.10 Layer creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALDatasetCreateLayer()* (C) or *Dataset.CreateLayer* (Python). The following layer creation options are supported:

- **OVERWRITE=[YES/NO]:** Defaults to `NO`. Whether to overwrite an existing table with the layer name to be created.
- **GEOMETRY_NULLABLE=[YES/NO]:** Defaults to `YES`. Whether the values of the geometry column can be NULL.
- **CARTODBFY=[YES/NO]:** Defaults to `YES`. Whether the created layer should be "Cartodbifi'ed" (i.e. registered in dashboard). Requires:
  - **SRS**: Output SRS must be EPSG:4326. You can use `-a_srs` or `-t_srs` to assign or transform to 4326 before importing.
  - **Geometry type**: Must be different than NONE. You can set to something generic with `-nlt GEOMETRY`.
- **LAUNDER=[YES/NO]:** Defaults to `YES`. This may be "YES" to force new fields created on this layer to have their field names "laundered" into a form more compatible with PostgreSQL. This converts to lower case and converts some special characters like "-" and "#" to "_". If "NO" exact names are preserved. If enabled the table (layer) name will also be laundered.

7.9.11 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). The following configuration options are available:

- **CARTO_API_URL=value:** Defaults to `https://[account_name].carto.com/api/v2/sql`. Can be used to point to another server.
- **CARTO_HTTPS=[YES/NO]:** can be set to NO to use http:// protocol instead of https:// (only if *CARTO_API_URL* is not defined).
- **CARTO_MAX_CHUNK_SIZE=value:**
- **CARTO_API_KEY=value:** see following paragraph.
- **CARTO_PAGE_SIZE=value:** Defaults to `500`. features are retrieved from the server by chunks of 500 by default. This number can be altered with the configuration option.

7.9.12 Examples

7.9.12.1 Example 1: Accessing data from a public table

```
ogrinfo -ro "Carto:gdalautotest2 tables=tm_world_borders_simpl_0_3"
```

7.9.12.2 Example 2: Creating and populating a table from a shapefile

```
ogr2ogr --config CARTO_API_KEY abcdefghijklmnopqrstuvw -f Carto "Carto:myaccount"␣
↪myshapefile.shp
```

7.9.12.3 Example 3: Creating and populating a table from a CSV containing geometries on EPSG:4326

```
ogr2ogr --config CARTO_API_KEY abcdefghijklmnopqrstuvw -f Carto "Carto:myaccount" file.
↪csv -a_srs 4326 -nlt GEOMETRY
```

Note

The `-a_srs` and `-nlt` must be provided to CARTODBFY since the information isn't extracted from the CSV.

7.9.13 See Also

- Carto API overview

7.10 Comma Separated Value (.csv)

Driver short name

CSV

Driver built-in by default

This driver is built-in by default

OGR supports reading and writing primarily non-spatial tabular data stored in text CSV files. CSV files are a common interchange format between software packages supporting tabular data and are also easily produced manually with a text editor or with end-user written scripts or programs.

The datasource name may be either a single CSV file or point to a directory. For a directory to be recognised as a .csv datasource at least half the files in the directory need to have the extension .csv. One layer (table) is produced from each .csv file accessed. Starting with GDAL 3.7, pipe separated values files with a ".psv" extension are also recognized.

For files structured as CSV, but not ending with the ".csv" extension, the 'CSV:' prefix can be added before the filename to force loading by the CSV driver.

Starting with GDAL 3.10, specifying the `-if CSV` option to command line utilities accepting it, or `CSV` as the only value of the `papszAllowedDrivers` of *`GDALOpenEx()`*, also forces the driver to recognize the passed filename, without the `CSV:` prefix.

The OGR CSV driver supports reading and writing. Because the CSV format has variable length text lines, reading is done sequentially. Reading features in random order will generally be very slow. OGR CSV layer might have a coordinate system stored in a .prj file (see GeoCSV specification). When reading a field named "WKT" is assumed to contain WKT geometry, but also is treated as a regular field. The OGR CSV driver returns all attribute columns as string data types if no field type information file (with .csvt extension) is available.

Limited type recognition can be done for Integer, Real, String, Date (YYYY-MM-DD), Time (HH:MM:SS+nn), DateTime (YYYY-MM-DD HH:MM:SS+nn) columns through a descriptive file with the same name as the CSV file, but a .csvt extension. In a single line the types for each column have to be listed with double quotes and be comma separated (e.g., "Integer","String"). It is also possible to specify explicitly the width and precision of each column, e.g. "Integer(5)","Real(10.7)","String(15)". The driver will then use these types as specified for the csv columns. Subtypes can be passed between parenthesis, such as "Integer(Boolean)", "Integer(Int16)" and "Real(Float32)". Accordingly with the GeoCSV specification, the "CoordX" or "Point(X)" type can be used to specify a column with longitude/easting values, "CoordY" or "Point(Y)" for latitude/northing values and "WKT" for geometries encoded in WKT

The "JSonStringList", "JSonIntegerList", "JSonInteger64List" and "JSonRealList" types can be used in .csvt to map to the corresponding OGR StringList, IntegerList, Integer64List and RealList types. The field values are then encoded as JSON arrays, with proper CSV escaping.

Automatic field type guessing can also be done if specifying the open options described in the below "Open options" section.

7.10.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.10.2 Format

CSV files have one line for each feature (record) in the layer (table). The attribute field values are separated by commas. At least two fields per line must be present. Lines may be terminated by a DOS (CR/LF) or Unix (LF) style line terminators. Each record should have the same number of fields. The driver will also accept a semicolon, a tabulation, a pipe, or a space character as field separator. Starting with GDAL 3.8, the autodetection will select the separator with the most occurrences if there are several candidates on the first line of the CSV file (and warn about that). The *SEPARATOR* open option may also be set to define the desired separator. Previous versions select comma by default when there are several potential separators.

Complex attribute values (such as those containing commas, quotes or newlines) may be placed in double quotes. Any occurrences of double quotes within the quoted string should be doubled up to "escape" them.

By default, the driver attempts to treat the first line of the file as a list of field names for all the fields. However, if one or more of the names is all numeric it is assumed that the first line is actually data values and dummy field names are generated internally (field_1 through field_n) and the first record is treated as a feature. Numeric values are treated as field names if they are enclosed in double quotes. This behavior can be modified via the HEADERS open option.

All CSV files are treated as UTF-8 encoded. A Byte Order Mark (BOM) at the beginning of the file will be parsed correctly. The layer creation option *WRITE_BOM* can be used to create a file with a Byte Order Mark, which can improve compatibility with some software (particularly Excel).

Example (employee.csv):

```
ID,Salary,Name,Comments
131,11000.0,Jane Lake,Chief Technical Officer
132,55000.0,John Walker,"The ""big"" cheese."
```

Note that the Comments value for the first data record is placed in double quotes because the value contains quotes, and those quotes have to be doubled up so we know we haven't reached the end of the quoted string yet.

Many variations of textual input are sometimes called Comma Separated Value files, including files without commas, but fixed column widths, those using tabs as separators or those with other auxiliary data defining field types or structure. This driver does not attempt to support all such files, but instead to support simple .csv files that can be auto-recognised. Scripts or other mechanisms can generally be used to convert other variations into a form that is compatible with the OGR CSV driver.

7.10.3 Reading CSV containing spatial information

7.10.3.1 Building point geometries

Consider the following CSV file (test.csv):

```
Latitude,Longitude,Name
48.1,0.25,"First point"
49.2,1.1,"Second point"
47.5,0.75,"Third point"
```

It is possible to directly specify the potential names of the columns that can contain X/longitude and Y/latitude with the *X_POSSIBLE_NAMES* and *Y_POSSIBLE_NAMES* open option.

ogrinfo -ro -al test.csv -oo X_POSSIBLE_NAMES=Lon -oo Y_POSSIBLE_NAMES=Lat* -oo KEEP_GEOM_COLUMNS=NO* will return :

```
OGRFeature(test):1
  Name (String) = First point
  POINT (0.25 48.1)

OGRFeature(test):2
  Name (String) = Second point
  POINT (1.1 49.2)

OGRFeature(test):3
  Name (String) = Third point
  POINT (0.75 47.5)
```

If the CSV file does not have a header line, the dummy "field_n" names can be used as possible names for coordinate fields. For example plain XYZ point data can be opened as

ogrinfo -ro -al elevation.xyz -oo X_POSSIBLE_NAMES=field_1 -oo Y_POSSIBLE_NAMES=field_2 -oo Z_POSSIBLE_NAMES=field_3

Otherwise, if one or several columns contain a geometry definition encoded as WKT, WKB (encoded in hexadecimal) or GeoJSON (in which case the GeoJSON content must be formatted to follow CSV rules, that is to say it must be surrounded by double-quotes, and double-quotes inside the string must be repeated for proper escaping), the name of such column(s) can be derived from the *GEOM_POSSIBLE_NAMES* open option.

For older versions, it is possible to extract spatial information (points) from a CSV file which has columns for the X and Y coordinates, through the use of the *VRT* driver.

You can write the associated VRT file (test.vrt):

```
<OGRVRTDataSource>
    <OGRVRTLayer name="test">
        <SrcDataSource>test.csv</SrcDataSource>
        <GeometryType>wkbPoint</GeometryType>
        <LayerSRS>WGS84</LayerSRS>
        <GeometryField encoding="PointFromColumns" x="Longitude" y="Latitude"/>
    </OGRVRTLayer>
</OGRVRTDataSource>
```

and *ogrinfo -ro -al test.vrt* will return :

```
OGRFeature(test):1
  Latitude (String) = 48.1
  Longitude (String) = 0.25
  Name (String) = First point
  POINT (0.25 48.1 0)

OGRFeature(test):2
  Latitude (String) = 49.2
  Longitude (String) = 1.1
  Name (String) = Second point
  POINT (1.1 49.200000000000003 0)

OGRFeature(test):3
  Latitude (String) = 47.5
  Longitude (String) = 0.75
  Name (String) = Third point
  POINT (0.75 47.5 0)
```

7.10.3.2 Building line geometries

Consider the following CSV file (test.csv):

```
way_id,pt_id,x,y
1,1,2,49
1,2,3,50
2,1,-2,49
2,2,-3,50
```

With a GDAL build with Spatialite enabled, *ogrinfo test.csv -dialect SQLite -sql "SELECT way_id, MakeLine(MakePoint(CAST(x AS float),CAST(y AS float))) FROM test GROUP BY way_id"* will return :

```
OGRFeature(SELECT):0
  way_id (String) = 1
  LINESTRING (2 49,3 50)

OGRFeature(SELECT):1
  way_id (String) = 2
  LINESTRING (-2 49,-3 50)
```

In fact we can get the same output with simply:

```
ogrinfo -sql "SELECT MakeLine(geometry) FROM test GROUP BY way_id" \
-oo X_POSSIBLE_NAMES=x -oo Y_POSSIBLE_NAMES=y -dialect SQLite test.csv
```

7.10.4 VSI Virtual File System API support

The driver supports reading and writing to files managed by VSI Virtual File System API, which include "regular" files, as well as files in the /vsizip/ (read-write) , /vsigzip/ (read-only) , /vsicurl/ (read-only) domains.

Writing to /dev/stdout or /vsistdout/ is also supported.

Reading from /vsistdin/ is supported using the `CSV:/vsistdin/` connection string and provided that none of the open options whose name starts with `AUTODETECT_` is used.

7.10.5 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are supported:

- **SEPARATOR=[AUTO/COMMA/SEMICOLON/TAB/SPACE/PIPE]:** (GDAL >= 3.8) Defaults to `AUTO`. Field separator character. Default value is AUTO for autodetection.
- **MERGE_SEPARATOR=[YES/NO]:** Defaults to `NO`. Setting it to YES will enable merging consecutive separators. Mostly useful when it is the space character.
- **AUTODETECT_TYPE=[YES/NO]:** Defaults to `NO`. Setting it to YES will enable auto-detection of field data types. If while reading the records (beyond the records used for autodetection), a value is found to not correspond to the autodetected data type, a warning will be emitted and the field will be emptied.
- **KEEP_SOURCE_COLUMNS=[YES/NO]:** Defaults to `NO`. keep a copy of the original columns where the guessing is active, and the guessed type is different from string. The name of the original columns will be suffixed with "_original". This flag should be used only when ..oo::*AUTODETECT_TYPE=YES*.
- **AUTODETECT_WIDTH=[YES/NO/STRING_ONLY]:** Defaults to `NO`. Setting it to YES to detect the width of string and integer fields, and the width and precision of real fields. Setting it to STRING_ONLY restricts to string fields. Setting it to NO select default size and width. If while reading the records (beyond the records used for autodetection), a value is found to not correspond to the autodetected width/precision, a warning will be emitted and the field will be emptied.
- **AUTODETECT_SIZE_LIMIT=<bytes>:** Defaults to `1000000`. size to specify the number of bytes to inspect to determine the data type and width/precision. The default will be 1 000 000. Setting 0 means inspecting the whole file. Note: when reading from standard input, this will be limited to 1 MB, due to how /vsistdin/ is implemented..
- **QUOTED_FIELDS_AS_STRING=[YES/NO]:** Defaults to `NO`. Only used if *`AUTODETECT_TYPE=YES`*. Whether to enforce quoted fields as string fields when set to YES. Otherwise, by default, the content of quoted fields will be tested for real, integer, etc... data types.
- **X_POSSIBLE_NAMES=<list_of_names>:** Comma separated list of possible names for X/longitude coordinate of a point. Each name might be a pattern using the star character in starting and/or ending position. E.g.: prefix*, *suffix or *middle*. The values in the column must be floating point values. *`X_POSSIBLE_NAMES`* and Y_POSSIBLE_NAMES must be both specified and a matching for each must be found in the columns of the CSV file. Only one geometry column per layer might be built when using *`X_POSSIBLE_NAMES`*/*`Y_POSSIBLE_NAMES`*.
- **Y_POSSIBLE_NAMES=<list_of_names>:** Comma separated list of possible names for Y/latitude coordinate of a point. Each name might be a pattern using the star character in starting and/or ending position. E.g.: prefix*, *suffix or *middle*. The values in the column must be floating point values. *`X_POSSIBLE_NAMES`* and *`Y_POSSIBLE_NAMES`* must be both specified and a matching for each must be found in the columns of the CSV file.
- **Z_POSSIBLE_NAMES=<list_of_names>:** Comma separated list of possible names for Z/elevation coordinate of a point. Each name might be a pattern using the star character in starting and/or ending position. E.g.: prefix*, *suffix or *middle*. The values in the column must be floating point values. Only taken into account in combination with *`X_POSSIBLE_NAMES`* and *`Y_POSSIBLE_NAMES`*.

- **GEOM_POSSIBLE_NAMES=<list_of_names>:** Comma separated list of possible names for geometry columns that contain geometry definitions encoded as WKT, WKB (in hexadecimal form, potentially in PostGIS 2.0 extended WKB) or GeoJSON. Each name might be a pattern using the star character in starting and/or ending position. E.g.: prefix*, *suffix or *middle*
- **KEEP_GEOM_COLUMNS=[YES/NO]:** Defaults to `YES`. Expose the detected X,Y,Z or geometry columns as regular attribute fields.
- **HEADERS=[YES/NO/AUTO]:** Defaults to `AUTO`. Whether the first line of the file contains column names or not. When set to AUTO, GDAL will assume the first line is column names if none of the values are strictly numeric.
- **EMPTY_STRING_AS_NULL=[YES/NO]:** Defaults to `NO`. Whether to consider empty strings as null fields on reading'.
- **MAX_LINE_SIZE=<integer>:** (GDAL >= 3.5.3) Defaults to `10000000`. Maximum number of bytes for a line (-1=unlimited).
- **OGR_SCHEMA=<filename>|<json string>:** (GDAL >= 3.11.0) Partially or totally overrides the auto-detected schema to use for creating the layer. The overrides are defined as a JSON list of field definitions. This can be a filename, a URL or JSON string conformant with the ogr_fields_override.schema.json schema This option takes precedence over any other option and over the .csvt file.

7.10.6 Creation Issues

The driver supports creating new databases (as a directory of .csv files), adding new .csv files to an existing directory or .csv files or appending features to an existing .csv table. Deleting or replacing existing features, or adding/modifying/deleting fields is supported, provided the modifications done are small enough to be stored in RAM temporarily before flushing to disk.

7.10.7 Layer Creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALDatasetCreateLayer()`* (C) or *`Dataset.CreateLayer`* (Python). The following layer creation options are supported:

- **LINEFORMAT=[CRLF/LF]:** By default when creating new .csv files they are created with the line termination conventions of the local platform (CR/LF on win32 or LF on all other systems). This may be overridden through use of the *`LINEFORMAT`* layer creation option which may have a value of **CRLF** (DOS format) or **LF** (Unix format).
- **GEOMETRY=[AS_WKT/AS_XYZ/AS_XY/AS_YX/NONE]:** Defaults to `NONE`. By default, the geometry of a feature written to a .csv file is discarded. It is possible to export the geometry in its WKT representation by specifying GEOMETRY=**AS_WKT**. It is also possible to export point geometries into their X,Y,Z components (different columns in the csv file) by specifying GEOMETRY=**AS_XYZ**, GEOMETRY=**AS_XY** or GEOMETRY=**AS_YX**. The geometry column(s) will be prepended to the columns with the attributes values. It is also possible to export geometries in GeoJSON representation using SQLite SQL dialect query, see example below.
- **CREATE_CSVT=[YES/NO]:** Defaults to `NO`. Create the associated .csvt file (see above paragraph) to describe the type of each column of the layer and its optional width and precision. This option also creates .prj file which stores coordinate system information.
- **SEPARATOR=[COMMA/SEMICOLON/TAB/SPACE/PIPE]:** Defaults to `COMMA`. Field separator character.
- **WRITE_BOM=[YES/NO]:** Defaults to `NO`. Write a UTF-8 Byte Order Mark (BOM) at the start of the file.
- **HEADER=[YES/NO]:** (GDAL >= 3.12) Defaults to `YES`. Whether to write a header line with the field names.
- **GEOMETRY_NAME=value:** Defaults to `WKT`. Name of geometry column. Only used if *`GEOMETRY=AS_WKT`* (and *`CREATE_CSVT=YES`* before GDAL 3.7.1).

- **STRING_QUOTING=[IF_NEEDED/IF_AMBIGUOUS/ALWAYS]:** Defaults to `IF_AMBIGUOUS`. whether to double-quote strings. IF_AMBIGUOUS means that string values that look like numbers will be quoted (it also implies IF_NEEDED). Defaults to IF_AMBIGUOUS (behavior in older versions was IF_NEEDED)

7.10.8 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The following configuration options are available:

- **OGR_WKT_PRECISION=<integer>:** Defaults to `15`. Number of decimals for coordinate values. A heuristic is used to remove insignificant trailing 00000x or 99999x that can appear when formatting decimal numbers. Examples: 6 gives 120.864, 24.1818; 2 gives 1.2E+02, 24.0.
- **OGR_WKT_ROUND=[YES/NO]:** Defaults to `YES`. Whether to enable the above mentioned heuristics to remove insignificant trailing 00000x or 99999x.

7.10.8.1 Examples

Example 1: Converting point shapefile to .csv with X,Y,Z coordinates columns

```
ogr2ogr output.csv input.shp -lco GEOMETRY=AS_XYZ
```

Example 2: Converting a shapefile to .csv with file with geometry field formatted as GeoJSON

```
ogr2ogr output.csv input.shp -dialect sqlite -sql \
    "select AsGeoJSON(geometry) AS geom, * from input"
```

Example 3: Converting a CSV into a GeoPackage

Specify the names of the coordinate columns and assign a coordinate reference system.

```
ogr2ogr \
  output.gpkg \
  input.csv \
  -oo X_POSSIBLE_NAMES=longitude \
  -oo Y_POSSIBLE_NAMES=latitude \
  -a_srs 'EPSG:4326'
```

Example 4: Densifying an input geometry specified in the WKT special field.

Note that one needs to specify the GEOMETRY=AS_WKT layer creation option, otherwise the input geometry would be returned unmodified.

```
$ cat input.csv
WKT,ID,Name
"LINESTRING (-900 -1450,-900 100)",0,900W

$ ogr2ogr -segmentize 400 -lco GEOMETRY=AS_WKT \
  -sql "SELECT ID, Name FROM input" output.csv input.csv

$ cat output.csv
WKT,ID,Name
"LINESTRING (-900 -1450,-900 -1062.5,-900 -675,-900 -287.5,-900 100)","0",900W
```

7.10.9 Particular datasources

The CSV driver can also read files whose structure is close to CSV files :

- Airport data files NfdcFacilities.xls, NfdcRunways.xls, NfdcRemarks.xls and NfdcSchedules.xls found on that FAA website
- Files from the USGS GNIS (Geographic Names Information System)
- The allCountries file from GeoNames
- Eurostat .TSV files

7.10.10 Other Notes

- GeoCSV specification
- Initial development of the OGR CSV driver was supported by DM Solutions Group and GoMOOS.
- Carto funded field type auto-detection and open options related to geometry columns.

7.11 CSW - OGC CSW (Catalog Service for the Web)

Driver short name

CSW

Build dependencies

libcurl

This driver can connect to a OGC CSW service. It supports CSW 2.0.2 protocol. GDAL/OGR must be built with Curl support in order to the CSW driver to be compiled. And the GML driver should be set-up for read support (thus requiring GDAL/OGR to be built with Xerces or Expat support).

It retrieves records with Dublin Core metadata.

7.11.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

7.11.2 Dataset name syntax

The minimal syntax to open a CSW datasource is : *CSW:* and the URL open option, or *CSW:http://path/to/CSW/endpoint*

7.11.3 Filtering

The driver will forward any spatial filter set with SetSpatialFilter() to the server. It also makes its best effort to do the same for attribute filters set with SetAttributeFilter() when possible (turning OGR SQL language into OGC filter description).

The *anytext* field can be queried to do a search in any text field. Note that we always return it as null content however in OGR side, to avoid duplicating information.

7.11.4 Issues

Some servers do not respect EPSG axis order, in particular latitude, longitude order for WGS 84 geodetic coordinates, so it might be needed to specify the *GML_INVERT_AXIS_ORDER_IF_LAT_LONG=NO* configuration option in those cases.

7.11.5 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are supported:

- **URL=value:** URL to the CSW server endpoint (if not specified in the connection string already)
- **ELEMENTSETNAME=[brief/summary/full]:** Defaults to `full`. Level of details of properties.
- **FULL_EXTENT_RECORDS_AS_NON_SPATIAL=[YES/NO]:** Defaults to `NO`. Whether records with (-180,-90,180,90) extent should be considered non-spatial.
- **OUTPUT_SCHEMA=value:** Value of outputSchema parameter, in the restricted set supported by the serve. Special value *gmd* can be used as a shortcut for http://www.isotc211.org/2005/gmd, *csw* for http://www.opengis.net/cat/csw/2.0.2. When this open option is set, a *raw_xml* field will be filled with the XML content of each record. Other metadata fields will remain empty.
- **MAX_RECORDS=value:** Defaults to `500`. Maximum number of records to retrieve in a single time. Servers might have a lower accepted value.

7.11.6 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). The following configuration options are available:

- **GML_INVERT_AXIS_ORDER_IF_LAT_LONG=[YES/NO]:** See *CRS support*. Equivalent of *INVERT_AXIS_ORDER_IF_LAT_LONG* open option.

7.11.7 Examples

7.11.7.1 Example 1: Listing all the records of a CSW server

```
ogrinfo -ro -al -noextent CSW:http://catalog.data.gov/csw
```

7.11.7.2 Example 2: Listing records of a CSW server matching a spatial and attribute filter:

```
ogrinfo -ro -al -noextent CSW:http://catalog.data.gov/csw -spat 2 49 2 49 -where
↪"subject LIKE '%mineralogy%'"
```

7.11.7.3 Example 3: Listing all the records of a CSW server that match a text on any text field

```
ogrinfo -ro -al -q CSW:http://catalog.data.gov/csw -spat 2 49 2 49 -where "anytext LIKE '
↪%France%'"
```

7.11.7.4 Example 4: Listing all the records of a CSW server as ISO 19115/19119

```
ogrinfo -ro -al -q CSW:http://catalog.data.gov/csw -oo OUTPUT_SCHEMA=gmd
```

7.11.8 See Also

- OGC CSW Standard
- *GML driver documentation*

7.12 Microstation DGN

Driver short name

DGN

Driver built-in by default

This driver is built-in by default

Microstation DGN files from Microstation versions predating version 8.0 are supported for reading (a *DGNv8 driver*, using Teigha libraries, is available to read and write DGN v8 files). The entire file is represented as one layer (named "elements").

DGN files are considered to have no georeferencing information through OGR. Features will all have the following generic attributes:

- Type: The integer type code as listed below in supported elements.
- Level: The DGN level number (0-63).
- GraphicGroup: The graphic group number.
- ColorIndex: The color index from the dgn palette.
- Weight: The drawing weight (thickness) for the element.
- Style: The style value for the element.
- EntityNum and MSLink: The Entity ID and MSLINK values in database linkage.
- ULink: User data linkage (multiple user data linkages may exist for each element).

DGN files do not contain spatial indexes; however, the DGN driver does take advantage of the extents information at the beginning of each element to minimize processing of elements outside the current spatial filter window when in effect.

7.12.1 Driver capabilities

Supports Create()

This driver supports the *`GDALDriver::Create()`* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.12.2 Supported Elements

The following element types are supported:

- Line (3): Line geometry.
- Line String (4): Multi segment line geometry.
- Shape (6): Polygon geometry.
- Curve (11): Approximated as a line geometry.
- B-Spline (21): Treated (inaccurately) as a line geometry.
- Arc (16): Approximated as a line geometry.
- Ellipse (15): Approximated as a line geometry.
- Text (17): Treated as a point geometry.

Generally speaking any concept of complex objects, and cells as associated components is lost. Each component of a complex object or cell is treated as a independent feature.

7.12.3 MSLINK

A DGN element can have a correspondence to a row in a database table, known as database linkage or database attribute. The EntityNum refers to the database table. The MSLink is the key to find the row in that table.

7.12.4 User data linkage

A DGN element may have multiple user data linkages. Each linkage has a user id, application id and a number of words of data. The user data linkage output reports the data for each different application id found as raw hexadecimal words (16bits). The application id is the second word of the raw data.

Is up to the user how to decode the user raw data, depending on the application id.

7.12.5 Styling Information

Some drawing information about features can be extracted from the ColorIndex, Weight and Style generic attributes; however, for all features an OGR style string has been prepared with the values encoded in ready-to-use form for applications supporting OGR style strings.

The various kinds of linear geometries will carry style information indicating the color, thickness and line style (i.e. dotted, solid, etc).

Polygons (Shape elements) will carry styling information for the edge as well as a fill color if provided. Fill patterns are not supported.

Text elements will contain the text, angle, color and size information (expressed in ground units) in the style string.

7.12.6 Creation Issues

2D DGN files may be written with OGR with significant limitations:

- Output features have the usual fixed DGN attributes. Attempts to create any other fields will fail.
- Virtual no effort is currently made to translate OGR feature style strings back into DGN representation information.
- POINT geometries that are not text (Text is NULL, and the feature style string is not a LABEL) will be translated as a degenerate (0 length) line element.
- Polygon, and multipolygon objects will be translated to simple polygons with all rings other than the first discarded.
- Polygons and line strings with too many vertices will be split into a group of elements prefixed with a Complex Shape Header or Complex Chain Header element as appropriate.
- A seed file must be provided (or if not provided, $PREFIX/share/gdal/seed_2d.dgn will be used). Many aspects of the resulting DGN file are determined by the seed file, and cannot be affected via OGR, such as initial view window.
- The various collection geometries other than MultiPolygon are completely discarded at this time.
- Geometries which fall outside the "design plane" of the seed file will be discarded, or corrupted in unpredictable ways.
- DGN files can only have one layer. Attempts to create more than one layer in a DGN file will fail.

7.12.7 Open options

Added in version 3.10.

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are supported:

- **ENCODING=<encoding>:** (GDAL >= 3.10) An encoding name supported by *CPLRecode()* (i.e. an iconv name) that indicates the encoding used by Text elements in the DGN file, to recode them to UTF-8. If not specified (or specified to UTF-8), no recoding will be done.

7.12.8 Dataset creation options

Dataset creation options can be specified in command-line tools using the syntax `-dsco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following dataset-creation options are supported:

- **3D=[YES/NO]:** Determine whether 2D (seed_2d.dgn) or 3D (seed_3d.dgn) seed file should be used. This option is ignored if the *SEED* option is provided.
- **SEED=<filename>:** Override the seed file to use.
- **COPY_WHOLE_SEED_FILE=[YES/NO]:** Defaults to `NO`. Indicate whether the whole seed file should be copied. If not, only the first three elements (and potentially the color table) will be copied.
- **COPY_SEED_FILE_COLOR_TABLE=[YES/NO]:** Defaults to `NO`. Indicates whether the color table should be copied from the seed file.
- **MASTER_UNIT_NAME=value:** Override the master unit name from the seed file with the provided one or two character unit name.
- **SUB_UNIT_NAME=value:** Override the sub unit name from the seed file with the provided one or two character unit name.

- **SUB_UNITS_PER_MASTER_UNIT=value:** Override the number of subunits per master unit. By default the seed file value is used.
- **UOR_PER_SUB_UNIT=value:** Override the number of UORs (Units of Resolution) per sub unit. By default the seed file value is used.
- **ORIGIN=<x,y,z>:** Override the origin of the design plane. By default the origin from the seed file is used.
- **ENCODING=<encoding>:** (GDAL >= 3.10) An encoding name supported by `CPLRecode()` (i.e. an iconv name) that indicates the encoding used by Text elements in the DGN file, to recode them from UTF-8. If not specified (or specified to UTF-8), no recoding will be done. "ISO-8859-1" (ISO Latin1) is supported even on builds without iconv support. Note that no fields in the DGN file itself contain the encoding name, hence it is the responsibility of the reader to open the file with the same value for the ENCODING open option.

- Dgnlib Page
- *Feature Style Specification*
- *DGNv8 driver* (using Teigha libraries)

7.13 Microstation DGN v8

Driver short name

DGNv8

Build dependencies

Open Design Alliance Teigha library

Microstation DGN files from Microstation version 8.0 are supported for reading and writing. Each model of the file is represented by a OGR layer.

This driver requires to be built against the (non open source) Open Design Alliance Teigha library.

DGN files are considered to have no georeferencing information through OGR. Features will all have the following generic attributes:

- Type: The integer type code as listed below in supported elements.
- Level: The DGN level number.
- GraphicGroup: The graphic group number.
- ColorIndex: The color index from the dgn palette.
- Weight: The drawing weight (thickness) for the element.
- Style: The style value for the element.
- ULink: User data linkage (multiple user data linkages may exist for each element).

7.13.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.13.2 Supported Elements

The following element types are supported in reading:

- Cell Header (2): used for polygons with holes
- Line (3): Line (2 points) geometry.
- Line String (4): Multi segment line geometry.
- Shape (6): Polygon geometry.
- TextNode (7): Container of Text elements.
- Curve (11): Approximated as a line geometry.
- ComplexString (12): Treated as line string or compound curve.
- ComplexShape (14): Treated as polygon or curve polygon.
- Ellipse (15): Approximated as a line geometry or a circular string.
- Arc (16): Approximated as a line geometry or a circular string.
- Text (17): Treated as a point geometry.
- B-Spline (21): Treated as a line geometry.
- PointString (22): Treated as multi point.
- Shared cell reference (35): Treated as point.

Generally speaking any concept of complex objects, and cells as associated components is lost. Each component of a complex object or cell is treated as a independent feature.

7.13.3 User data linkage

A DGN element may have multiple user data linkages. Each linkage has a user id, application id and a number of words of data. The user data linkage output reports the data for each different application id.

For unknown application ids, the raw data is reported as hexadecimal words (16bit). Is up to the user how to decode the user data, depending on the application id.

7.13.4 Styling Information

Some drawing information about features can be extracted from the ColorIndex, Weight and Style generic attributes; however, for all features an OGR style string has been prepared with the values encoded in ready-to-use form for applications supporting OGR style strings.

The various kinds of linear geometries will carry style information indicating the color, thickness and line style (i.e. dotted, solid, etc).

Polygons (Shape elements) will carry styling information for the edge as well as a fill color if provided. Fill patterns are not supported.

Text elements will contain the text, angle, color and size information (expressed in ground units) in the style string.

7.13.5 Metadata

The various metadata items that can be set in the DGN header with the dataset creation options (see below) can be retrieved in the "DGN" metadata domain.

7.13.6 Creation Issues

DGN files may be written with OGR with limitations:

- Output features have the usual fixed DGN attributes. Attempts to create any other fields will fail.
- Translation from OGR feature style strings back into DGN representation information is limited to a few properties of LABEL (text, font name, size, angle, color), PEN (color) and BRUSH (fill color) tools.
- POINT geometries that are not text (Text is NULL, and the feature style string is not a LABEL) will be translated as a degenerate (0 length) line element.
- Geometries which fall outside the "design plane" of the seed file will be discarded, or corrupted in unpredictable ways.

7.13.7 Dataset creation options

Dataset creation options can be specified in command-line tools using the syntax `-dsco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following dataset creation options are supported:

- **SEED=<filename>:** Specify the seed file to use.
- **COPY_SEED_FILE_COLOR_TABLE=[YES/NO]:** Defaults to `NO`. Indicates whether the color table should be copied from the seed file. Only taken into account if *`SEED`* is specified.
- **COPY_SEED_FILE_MODEL=[YES/NO]:** Defaults to `YES`. Indicates whether the existing models (without their graphic contents) should be copied from the seed file. This holds as well for the view groups and named views to which they are linked to. Only taken into account if *`SEED`* is specified.
- **COPY_SEED_FILE_MODEL_CONTROL_ELEMENTS=[YES/NO]:** Defaults to `YES`. Indicates whether the existing control elements of models should be copied from the seed file. Only taken into account if *`COPY_SEED_FILE_MODEL=YES`*.
- **APPLICATION=value:** Set Application field in header. If not specified, derived from seed file when set. Otherwise mentions the version of GDAL and the Teigha library used.
- **TITLE=value:** Set Title field in header. If not specified, from the seed file.
- **SUBJECT=value:** Set Subject field in header. If not specified, from the seed file.
- **AUTHOR=value:** Set Author field in header. If not specified, from the seed file.
- **KEYWORDS=value:** Set Keywords field in header. If not specified, from the seed file.

- **TEMPLATE=value:** Set Template field in header. If not specified, from the seed file.
- **COMMENTS=value:** Set Comments field in header. If not specified, from the seed file.
- **LAST_SAVED_BY=value:** Set LastSavedBy field in header. If not specified, from the seed file.
- **REVISION_NUMBER=value:** Set RevisionNumber field in header. If not specified, from the seed file.
- **CATEGORY=value:** Set Category field in header. If not specified, from the seed file.
- **MANAGER=value:** Set Manager field in header. If not specified, from the seed file.
- **COMPANY=value:** Set Company field in header. If not specified, from the seed file.

7.13.8 Layer creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALDatasetCreateLayer()`* (C) or *`Dataset.CreateLayer`* (Python). The following layer creation options are supported:

- **DESCRIPTION=value:** Description associated with the layer. If not specified, from the seed file.
- **DIM=[2/3]:** Defaults to `3`. Dimension (ie 2D vs 3D) of the layer. By default, 3, unless the model is reused from the seed file.

7.13.9 Building

See *ODA platform support* for building GDAL with ODA support.

7.13.9.1 Open Design Alliance - ODA Platform

ODA Platform (previously named Teigha) is required to enable GDAL support for reading AutoCAD DWG and Microstation DGN v8 files. GDAL/OGR must be built with ODA support in order to enable these drivers.

ODA required products

ODA Platform includes several SDK. Drawings SDK provides access to all data in .dwg and .dgn through an object-oriented API. It is required to compile GDAL. Since Kernel SDK is required by all products, these two products must be downloaded:

- Kernel
- Drawings

These libraries are not publicly available. You have to became a member to get access to the libraries. Upon authentication the libraries are available from: ODA Member Downloads

Get the libraries

To selected the appropriate files to download, consider the following ODA name conventions (for Linux):

- lnx - Linux
- X86, X64 - indicates X86 or X64 platform
- 4.4, 4.7, 4.8, 4.9, 5.2, 5.3, 6.3, 7.2, 8.3 - GCC versions
- dll - indicates a shared library version
- pic - compiled with Position Independent Code option

ODA archives also contains a release suffix in order to distinguish between releases, like 21.2 or 21.6.

To download the required files for Linux, the following files could be downloaded:

- *Kernel_lnxX64_7.2dll_21.6.tar.gz*
- *Drawings_lnxX64_7.2dll_21.6.tar.gz*

In this example, the files names are:

- *lnx* for Linux
- *X64* for X64 architecture
- *7.2* for gcc 7.2
- *dll* for shared library version
- *21.6* ODA 2021 release, build 6

Compiling the libraries

The libraries must be merged before compiling.

```
cd ~/dev/cpp/ODA21.6
mkdir base_7.2
tar xvzf Kernel_lnxX64_7.2dll_21.6.tar.gz -C base_7.2
tar xvzf Drawings_lnxX64_7.2dll_21.6.tar.gz -C base_7.2
```

To compile, an activation key is required. It can be requested from ODA Products Activation. The activation key must be copied to *ThirdParty/activation/*.

```
cp OdActivationInfo base_7.2/ThirdParty/activation/
```

Compile the ODA libraries with:

```
cd base_7.2
./configure
make -j8
```

Make sure your gcc major version matches ODA libs gcc version. On Ubuntu, for example, you can install different gcc/g++ versions, like 7, 8 and 9. Switch between them with:

```
sudo update-alternatives --config gcc
sudo update-alternatives --config g++
```

ODA libraries path

After compiling ODA, the resulting libs are in a non standard search path. There is no *make install* included to copy the libraries to a standard location. This might be an issue.

You have different alternative options to compile and run GDAL/OGR with ODA:

- copy the ODA libraries to a standard location
- set LD_LIBRARY_PATH (like *LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/jgr/dev/cpp/ODA21.6/base_7.2/bin/lnxX64_7*
- Adding ODA folder to the system library path (*echo "/home/jgr/dev/cpp/ODA21.6/base_7.2/bin/lnxX64_7.2dll" | sudo tee -a /etc/ld.so.conf.d/z_gdal-ODA.conf*)
- Setting a run time path (*rpath*) when compiling GDAL (like *LDFLAGS="-Wl,-rpath=/home/jgr/dev/cpp/ODA21.6/base_7.2/bin/lnxX64_7.2dll"*).

ODA library names

Some ODA library names do not conform the usual Linux standard *lib*.so*. If you don't use *rpath*, for the other alternatives listed above, you might have to create symbolic links from the actual names. Example:

```
cd ~/dev/cpp/ODA21.6/base_7.2/bin/lnxX64_7.2dll
for f in *.tx
do
   echo "Processing $f"
   ln -s $f lib$f.so
done
sudo ldconfig
```

Check with *ldconfig -v* if all ODA libraries are now visible.

Compiling GDAL

After compiling ODA libraries, GDAL can be build using two additional options:

- *--with-teigha=/home/jgr/dev/cpp/ODA21.6/base_7.2*
- *--with-teigha-plt=lnxX64_7.2dll*

The value of *--with-teigha* is the full path of the folder where the libraries were merged and compiled.

The value of *--with-teigha-plt* must match the platform name used by ODA. The platform name is the folder name under *Platforms*:

```
ls -l Platforms/
lnxX64_7.2dll
```

GDAL compilation

Use your own GDAL build configuration and add the previous mentioned options. After running *configure*, make sure that *Teigha (DWG and DGNv8)* support is configured.

As an example, compiling GDAL can be:

```
cd gdal
./autogen.sh
./configure --without-libtool LDFLAGS="-L/usr/lib/x86_64-linux-gnu" --with-
↪python=python3 --with-proj=/usr/local --with-pg=yes --with-poppler --with-teigha=/home/
↪jgr/dev/cpp/ODA21.6/base_7.2 --with-teigha-plt=lnxX64_7.2dll
make -j8
sudo make install
sudo ldconfig
# Python support
cd swig/python
python3 setup.py build
sudo python3 setup.py install
```

We added *LDFLAGS="-L/usr/lib/x86_64-linux-gnu"* to use system libs over ODA's *libpcre*, *libcurl*, etc.

Testing

After compiling GDAL, you can check if the new drivers *DGNV8* and *DWG* are supported with:

```
./apps/ogrinfo --formats | grep 'AutoCAD\|Microstation'
DGN -vector- (rw+v): Microstation DGN
DWG -vector- (ro): AutoCAD DWG
DGNV8 -vector- (rw+): Microstation DGNv8
DXF -vector- (rw+v): AutoCAD DXF
CAD -raster,vector- (rovs): AutoCAD Driver
```

If a file is DGNv8, you will see that driver in action when opening the file:

```
ogrinfo ~/dev/cpp/gdal/autotest/ogr/data/dgnv8/test_dgnv8.dgn
INFO: Open of `/home/jgr/dev/cpp/gdal/autotest/ogr/data/dgnv8/test_dgnv8.dgn'
      using driver `DGNV8' successful.
1: my_model
```

Troubleshooting

If you find linking errors, you can set *LD_LIBRARY_PATH* or *LDFLAGS* environment variables to make sure you are able to get the ODA libraries from their location.

Use *ldconfig -v* to check if ODA's library folder is listed.

For example, you can try:

```
export LD_LIBRARY_PATH=/home/jgr/dev/cpp/ODA21.6/base_7.2/bin/lnxX64_7.2dll
./configure --without-libtool LDFLAGS="-L/usr/lib/x86_64-linux-gnu" --with-
↪python=python3 --with-proj=/usr/local --with-pg=yes --with-poppler --with-teigha=/home/
↪jgr/dev/cpp/ODA21.6/base_7.2 --with-teigha-plt=lnxX64_7.2dll
```

You can force a run time location (with *rpath*) with:

```
./configure --without-libtool LDFLAGS="-L/usr/lib/x86_64-linux-gnu -Wl,-rpath=/home/jgr/
↪dev/cpp/ODA21.6/base_7.2/bin/lnxX64_7.2dll" --with-python=python3 --with-proj=/usr/
↪local --with-pg=yes --with-poppler --with-teigha=/home/jgr/dev/cpp/ODA21.6/base_7.2 --
↪with-teigha-plt=lnxX64_7.2dll
```

Adjust these settings, according to your build environment.

See Also

- Introducing the ODA Platform
- *AutoCAD DWG*
- *Microstation DGN v8*

- *DGN (v7) driver*
- *Feature Style Specification*

7.14 AutoCAD DWG

Driver short name

DWG

Build dependencies

Open Design Alliance Teigha library

OGR supports reading most versions of AutoCAD DWG when built with the Open Design Alliance Teigha library. DWG is a binary working format used for AutoCAD drawings. A reasonable effort has been made to make the OGR DWG driver work similarly to the OGR DXF driver which shares a common data model. The entire contents of the .dwg file is represented as a single layer named "entities".

DWG files are considered to have no georeferencing information through OGR. Features will all have the following generic attributes:

- Layer: The name of the DXF layer. The default layer is "0".
- SubClasses: Where available, a list of classes to which an element belongs.
- ExtendedEntity: Where available, extended entity attributes all appended to form a single text attribute.
- Linetype: Where available, the line type used for this entity.
- EntityHandle: The hexadecimal entity handle. A sort of feature id.
- Text: The text of labels.

A reasonable attempt is made to preserve line color, line width, text size and orientation via OGR feature styling information when translating elements. Currently no effort is made to preserve fill styles or complex line style attributes.

The *OGR_ARC_STEPSIZE* and *OGR_ARC_MAX_GAP* configurations options control the approximation of arcs, ellipses, circles and rounded polylines as linestrings.

7.14.1 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). The following configuration options are available:

- **DWG_INLINE_BLOCKS=[TRUE/FALSE]:** The default behavior is for block references to be expanded with the geometry of the block they reference. However, if the *DWG_INLINE_BLOCKS* configuration option is set to the value FALSE, then the behavior is different as described here:
  - A new layer will be available called blocks. It will contain one or more features for each block defined in the file. In addition to the usual attributes, they will also have a BlockName attribute indicate what block they are part of.
  - The entities layer will have new attributes BlockName, BlockScale, BlockAngle and BlockAttributes.
  - BlockAttributes will be a list of (tag x value) pairs of all visible attributes (JSON encoded).
  - block referenced will populate these new fields with the corresponding information (they are null for all other entities).
  - block references will not have block geometry inlined - instead they will have a point geometry for the insertion point.

The intention is that with *DWG_INLINE_BLOCKS* disabled, the block references will remain as references and the original block definitions will be available via the blocks layer.

- **DWG_ATTRIBUTES=[TRUE/FALSE]:** If option is set to TRUE value, then block attributes are treated as feature attributes, one feature attribute for each tag. This option allow conversion to rows and columns data such as database tables.
- **DWG_ALL_ATTRIBUTES=[TRUE/FALSE]:** Defaults to `TRUE`. If option is set to FALSE value, then block attributes are ignored if the visible property of the tag attribute is false. To see all attributes set *DWG_ALL_ATTRIBUTES* to TRUE value (this is the default value).
- **DWG_CLOSED_LINE_AS_POLYGON=[TRUE/FALSE]:** (GDAL >= 3.10) Defaults to `FALSE`. This option can be set to TRUE specified to ask for closed POLYLINE and LWPOLYLINE to be exposed as OGR polygons.

7.14.2 Building

See *ODA platform support* for building GDAL with ODA support.

7.15 AutoCAD DXF

Driver short name

DXF

Driver built-in by default

This driver is built-in by default

DXF is an ASCII format used for interchanging AutoCAD drawings between different software packages. OGR supports reading DXF files generated by all recent versions of AutoCAD, and writing DXF files that are compatible with AutoCAD 2004 and later.

Starting with GDAL 3.12, the DXF driver can also read the AutoCAD Binary DXF format (not to be confused with DWG!)

DXF files are considered to have no georeferencing information through OGR.

7.15.1 Driver capabilities

Supports Create()

This driver supports the *`GDALDriver::Create()`* operation

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.15.2 DXF Reader

By default, the entire contents of the file are represented as a single OGR layer named "entities". Features will all have the following generic fields:

- Layer: The name of the DXF layer. The default layer is "0".
- PaperSpace: 1 if the entity is located on a layout (paper space), NULL otherwise.
- SubClasses: Where available, a list of classes to which an entity belongs.
- RawCodeValues: A string list containing all group codes and values that are not handled by the DXF reader. Only available when the configuration option *DXF_INCLUDE_RAW_CODE_VALUES=TRUE*.

  DXF_INCLUDE_RAW_CODE_VALUES=[TRUE/FALSE]: Defaults to FALSE.

  - Linetype: Where available, the line type used for this entity.
- EntityHandle: The hexadecimal entity handle. A sort of feature id.
- Text: The text of labels.

7.15.2.1 Supported Entities

The following entity types are supported:

- POINT: Produces a simple POINT geometry feature.
- MTEXT, TEXT: Produces a POINT feature with LABEL style information. The style string may include the following parameters: f s t a c w p dx dy bo it. Text positioning might not precisely match CAD software (especially the vertical alignment of MTEXT), as calculation of the exact position would require GDAL to be aware of the font metrics. By default, character escapes such as %%p are honored where applicable, and MTEXT control sequences like \Wx.xx; are stripped. To disable this behavior and retrieve the raw text values, set the configuration option *DXF_TRANSLATE_ESCAPE_SEQUENCES=FALSE*.

  DXF_TRANSLATE_ESCAPE_SEQUENCES=[TRUE/FALSE]: Defaults to TRUE.
- LINE, POLYLINE, LWPOLYLINE: Translated as a LINESTRING. Rounded polylines (those with their vertices' bulge attributes set) will be tessellated. Single-vertex polylines are translated to POINT. Polyface meshes are translated as POLYHEDRALSURFACE geometries.

  Starting with GDAL 3.10, the *DXF_CLOSED_LINE_AS_POLYGON* configuration option can be set to TRUE to ask for closed POLYLINE and LWPOLYLINE to be exposed as OGR polygons.

  DXF_CLOSED_LINE_AS_POLYGON=[TRUE/FALSE]: Defaults to FALSE.
- MLINE: Translated as a MULTILINESTRING. Only the geometry is reconstructed; styling applied to individual line elements within the MLINE is ignored. Fill colors and start/end caps are also omitted.
- CIRCLE, ELLIPSE, ARC, SPLINE,HELIX: Translated as a LINESTRING, tessellating the curve into line segments.

  CIRCLEs with nonzero "thickness" (cylinders) are approximated as a POLYHEDRALSURFACE.
- INSERT: By default, the block definition referenced by the INSERT will be inserted as a compound geometry (for example, a MULTILINESTRING for a block containing many lines, or a GEOMETRYCOLLECTION for a block that contains points and lines). If the block contains TEXT or MTEXT entities, they are not merged into the compound geometry and are instead returned as separate features.

  Three configuration options are available to control the behavior of INSERT entities:

  - **DXF_MERGE_BLOCK_GEOMETRIES=[TRUE/FALSE]:** Defaults to TRUE. To avoid merging blocks into a compound geometry the *DXF_MERGE_BLOCK_GEOMETRIES* config option may be set to FALSE. Use this option if you need to preserve the styling (such as colors) of individual linework entities within the block.
  - **DXF_INLINE_BLOCKS=value:** See below.
  - **DXF_FEATURE_LIMIT_PER_BLOCK=value:** Defaults to 10000. Maximum number of features inserted from a single block. Set to -1 for no limit.

- ATTDEF, ATTRIB: Attributes (ATTRIB) are treated as TEXT entities, and attribute definitions (ATTDEF) inside blocks are ignored. The behavior is different when *DXF_INLINE_BLOCKS* is false (see below).
- HATCH: Line and arc boundaries are collected as a polygon geometry, but no effort is currently made to represent the fill style of HATCH entities. Behavior is affected by the following configuration option:
  - **DXF_HATCH_TOLERANCE=value:** Determines the tolerance used when looking for the next component to add to the hatch boundary.
- 3DFACE, SOLID, TRACE: Translated as POLYGON, except for SOLID and TRACE entities with only one distinct vertex (translated as POINT) or two distinct vertices (translated as LINESTRING).
- DIMENSION: The DXF format allows each DIMENSION entity to reference an "anonymous" block (a block whose name starts with *D) that contains the geometry of the DIMENSION. If present, this anonymous block will be inlined at the required position. Otherwise, fallback will occur to a simple DIMENSION renderer that explodes a linear dimension as a MULTILINESTRING feature. Arrowheads, if present, are translated as one or more additional features. The fallback renderer will render nonlinear dimensions as if they were linear.
- LEADER, MULTILEADER: The leader line is translated as a LINESTRING (LEADER) or MULTILINESTRING (MULTILEADER). Arrowheads, if present, are translated as one or more additional features. Text for MULTILEADER entities is translated into a POINT feature with a label. Block content for MULTILEADERS is treated as for INSERT. Spline leaders are tessellated into line segments.
- WIPEOUT: a light support of WIPEOUT entities can parse the outline geometry of WIPEOUT entities and translate it into a POLYGON feature. The feature is 2D support only - Z coordinates are ignored.
- 3DSOLID, REGION, BODY, SURFACE: See below.

A reasonable attempt is made to preserve color, line width (lineweight), line type, text size and orientation via OGR feature styling information when translating entities. Currently no effort is made to preserve complex line types (those that include text or shapes) or HATCH fill styles.

The *OGR_ARC_STEPSIZE* and *OGR_ARC_MAX_GAP* configurations options control the approximation of arcs, ellipses, circles and rounded polylines as linestrings.

For splines, the interpolated polyline contains eight vertices for each control point.

Object coordinate systems (OCS), also known as "extrusions", are supported for all entities to which they apply as per the DXF specification, except DIMENSION, LEADER and MULTILEADER. These three entity types also currently lack support for elevations; the geometries will always be 2D.

7.15.2.2 DXF_INLINE_BLOCKS

The default behavior is for INSERT entities to be exploded with the geometry of the BLOCK they reference. However, if the *DXF_INLINE_BLOCKS* configuration option is set to the value FALSE, then the behavior is different as described here.

- A new layer will be available called "blocks". It will contain one or more features for each BLOCK defined in the file. In addition to the usual fields, they will also have a Block field indicating what block they are part of.
- ATTDEF entities in the blocks layer will have an AttributeTag field, giving the tag of the ATTDEF entity.
- The entities layer will have several new fields:
  - BlockName: The name of the referenced block.
  - BlockScale: The X, Y, and Z scale factors.
  - BlockAngle: The angle of the block in degrees.
  - BlockOCSNormal: The unit normal vector of the object coordinate system (OCS) of the INSERT entity.
  - BlockOCSCoords: The OCS coordinates of the insertion point.

  - BlockAttributes: The text content of attributes associated with this block. Each entry in this string list contains an attribute tag, followed by a space, followed by the text for that attribute (which may be empty).

- INSERT entities will populate these new fields with the corresponding information (they are null for all other entities).
- INSERT entities will not have block geometry inlined - instead they will have a POINT geometry for the insertion point.

The intention is that with *DXF_INLINE_BLOCKS* disabled, the block references will remain as references and the original block definitions will be available via the blocks layer. On export this configuration will result in the creation of similar blocks.

7.15.2.3 3D Extensibility

DXF files may contain 3DSOLID, REGION, BODY and SURFACE entities, which contain 3D modelling data in the proprietary Autodesk ShapeManager (ASM) format, a broadly compatible fork of the ACIS format. GDAL cannot transform these entities into OGR geometries, so they are skipped by default.

This behavior can be controlled with the following configuration option:

- **DXF_3D_EXTENSIBLE_MODE=[TRUE/FALSE]:** If TRUE, include ASM entities with the raw ASM data stored in a field, allowing for interoperability with commercial conversion tools. This option adds two new fields:
  - ASMData: A binary field that contains the ASM data.
  - ASMTransform: A column-major list of 12 real values indicating the affine transformation to be applied to the entity.

This feature only works for DXF files in AutoCAD 2013 (AC1027) format and later.

7.15.2.4 Character Encodings

Normally DXF files are in the ANSI_1252 / Win1252 encoding. GDAL/OGR attempts to translate this to UTF-8 when reading and back into ANSI_1252 when writing. DXF files can also have a header field ($DWGCODEPAGE) indicating the encoding of the file. An attempt is made to use this to recode other code pages to UTF-8. Whether this works will depend on the code page naming and whether GDAL/OGR is built against the iconv library for character recoding.

In some cases the $DWGCODEPAGE setting in a DXF file will be wrong, or unrecognised by OGR. In this case, *DXF_ENCODING* may be used to override what id will be used by OGR in transcoding:

- **DXF_ENCODING=value:** An encoding name supported by *CPLRecode()* (i.e. an iconv name), not a DXF $DWGCODEPAGE name. Using a value "UTF-8" will avoid any attempt to recode the text as it is read.

7.15.3 Open options

Added in version 3.10.

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are supported:

- **CLOSED_LINE_AS_POLYGON=[YES/NO]:** (GDAL >= 3.10) Defaults to `NO`. See *DXF_CLOSED_LINE_AS_POLYGON*
- **INLINE_BLOCKS=[YES/NO]:** (GDAL >= 3.10) Defaults to `YES`. See *DXF_INLINE_BLOCKS*
- **MERGE_BLOCK_GEOMETRIES=[YES/NO]:** (GDAL >= 3.10) Defaults to `YES`. See *DXF_MERGE_BLOCK_GEOMETRIES*
- **TRANSLATE_ESCAPE_SEQUENCES=[YES/NO]:** (GDAL >= 3.10) Defaults to `YES`. See *DXF_TRANSLATE_ESCAPE_SEQUENCES*

- **INCLUDE_RAW_CODE_VALUES=[YES/NO]:** (GDAL >= 3.10) Defaults to NO. See *DXF_INCLUDE_RAW_CODE_VALUES*
- **3D_EXTENSIBLE_MODE=[YES/NO]:** (GDAL >= 3.10) Defaults to NO. See *DXF_3D_EXTENSIBLE_MODE*
- **HATCH_TOLERANCE=<tolerance>:** (GDAL >= 3.10) See *DXF_HATCH_TOLERANCE*
- **ENCODING=<encoding>:** (GDAL >= 3.10) See *DXF_ENCODING*

7.15.4 DXF Writer

DXF files are written in AutoCAD 2004 format. A standard header (everything up to the ENTITIES keyword) is written from the $GDAL_DATA/header.dxf file, and the $GDAL_DATA/trailer.dxf file is added after the entities. Only one OGR layer can be used to create the output file (but many DXF layers can be created - see below).

- Point features with LABEL styling are written as MTEXT entities based on the styling information.
- Point features without LABEL styling are written as POINT entities.
- LineString and MultiLineString features are written as one or more LWPOLYLINE entities, closed in the case of polygon rings. If the geometry does not have a constant elevation, a POLYLINE entity is written. An effort is made to preserve line width and color.
- Polygon, Triangle and MultiPolygon features are written as HATCH entities by default. To write these features as LWPOLYLINE/POLYLINE entities instead, set the configuration option *DXF_WRITE_HATCH=FALSE*:
  - **DXF_WRITE_HATCH=[TRUE/FALSE]:** Defaults to TRUE. If TRUE, write Polygon, Triangle, and MultiPolygon features as HATCH entities. If false, write them as LWPOLYLINE/POLYLINE.

  You may need to do this if your geometries do not have a constant elevation, as the DXF HATCH entity cannot represent such geometries.

Only the first tool (PEN, BRUSH, etc) in the style string is read. The following style string parameters are understood:

Tool	Available on geometry types	Supported parameters
PEN	Point, (Multi)LineString	color (c); width (w); dash pattern (p)
BRUSH	(Multi)Polygon, Triangle	foreground color (fc), background color (bc), brush id (id), angle (a), scaling factor (s)
SYMBOL	Point	color (c)
LABEL	Point	text (t); font name (f); font size (s), treated as cap height; bold (bo); italic (it); text color (c); x and y offsets (dx, dy); angle (a); anchor point (p); stretch (w)

Note: for BRUSH, only foreground color is supported before GDAL 3.12.

Dataset creation options can be specified in command-line tools using the syntax `-dsco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The driver supports the following dataset creation options:

- **HEADER=<filename>:** Override the header file used - in place of header.dxf located in the GDAL_DATA directory.
- **TRAILER=<filename>:** Override the trailer file used - in place of trailer.dxf located in the GDAL_DATA directory.
- **FIRST_ENTITY=<integer>:** Identifier of first entity

- **INSUNITS=[AUTO/HEADER_VALUE/UNITLESS/INCHES/FEET/MILLIMETERS/CENTIMETERS /METERS/US_SURVEY_FEET]:** (GDAL >= 3.11) Defaults to `AUTO`. Drawing units for the model space ($INSUNITS system variable). The default `AUTO` mode first check if the written layer has a projected CRS, and if so uses is linear units to determine the value of `$INSUNITS`. Otherwise it fallbacks to the value of the header template (`HEADER_VALUE` mode), which is `INCHES`.
- **MEASUREMENT=[HEADER_VALUE/IMPERIAL/METRIC]:** (GDAL >= 3.11) Defaults to `HEADER_VALUE`. Whether the current drawing uses imperial or metric hatch pattern and linetype ($MEASUREMENT system variable). Defaults to the value of the header template, which is `IMPERIAL`.

The header and trailer templates can be complete DXF files. The driver will scan them and only extract the needed portions (portion before or after the ENTITIES section).

7.15.4.1 Block References

It is possible to export a "blocks" layer to DXF in addition to the "entities" layer in order to produce actual DXF BLOCKs definitions in the output file. It is also possible to write INSERT entities if a block name is provided for an entity. To make this work the following conditions apply:

- A "blocks" layer may be created, and it must be created before the entities layer.
- The entities in the blocks layer should have the Block field populated.
- Objects to be written as INSERTs in the entities layer should have a POINT geometry, and the BlockName field set. You may also set BlockAngle, BlockScale, BlockOCSNormal and BlockOCSCoords (see above under *DXF_INLINE_BLOCKS* for details). If BlockOCSCoords is set to a list of 3 real numbers, it is used as the location of the block; in this situation the position of the POINT geometry is ignored.
- If a block (name) is already defined in the template header, that will be used regardless of whether a new definition was provided in the blocks layer.

The intention is that a simple translation from DXF to DXF with *DXF_INLINE_BLOCKS* set to FALSE will approximately reproduce the original blocks and keep INSERT entities as INSERT entities rather than exploding them.

7.15.4.2 Layer Definitions

When writing entities, if populated the Layer field is used to set the written entities layer. If the layer is not already defined in the template header then a new layer definition will be introduced, copied from the definition of the default layer ("0").

7.15.4.3 Linetype Definitions

When writing linestring geometries, the following rules apply with regard to linetype (dash pattern) definitions.

- If the Linetype field is set on a written feature, and that linetype is already defined in the template header, then it will be referenced from the entity. If a style string is present with a "p" pattern proportional to the linetype defined in the header, a linetype scale value is written.
- If the Linetype field is set, but the linetype is not defined in the header template, then a definition will be added if the feature has an OGR style string with a PEN tool and a "p" pattern setting.
- If the feature has no Linetype field set, but it does have an OGR style string with a PEN tool with a "p" pattern set, then an automatically named linetype will be created in the output file. Or, if an appropriate linetype was previously created, that linetype will be referenced, with a linetype scale if required.

The intention is that "dot dash" style patterns will be preserved when written to DXF and that specific linetypes can be predefined in the header template, and referenced using the Linetype field if desired.

It is assumed that patterns are using "g" (georeferenced) units for defining the line pattern. If not, the scaling of the DXF patterns is likely to be wrong - potentially very wrong.

7.15.5 Binary DXF to ASCII DXF direct translation

Added in version 3.12.

A command like `ogr2ogr out_ascii.dxf in_binary.dxf` without any additional parameter will go through a direct conversion path.

7.15.6 See also

List of known issues KNOWN_ISSUES.md

AutoCAD 2000 DXF Reference

AutoCAD 2014 DXF Reference

7.16 Google Earth Engine Data API

Driver short name

EEDA

Build dependencies

libcurl

The driver supports read-only operations to list images and their metadata as a vector layer, using Google Earth Engine REST API.

7.16.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

7.16.2 Dataset name syntax

The minimal syntax to open a datasource is:

```
EEDA:[collection]
```

where collection is something like projects/earthengine-public/assets/COPERNICUS/S2.

7.16.3 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are available:

- **COLLECTION=value:** To specify the collection if not specified in the connection string.
- **VSI_PATH_FOR_AUTH=value:** (GDAL >= 3.10) /vsigs/... path onto which a GOOGLE_APPLICATION_CREDENTIALS path specific option is set

7.16.4 Authentication methods

The following authentication methods can be used:

- Authentication Bearer header passed through the EEDA_BEARER or *EEDA_BEARER_FILE* configuration options.
- Service account private key file, through the *GOOGLE_APPLICATION_CREDENTIALS* configuration option, or set as a path-specific option whose value is set in the VSI_PATH_FOR_AUTH open option.
- OAuth2 Service Account authentication through the *EEDA_PRIVATE_KEY*/ *EEDA_PRIVATE_KEY_FILE* + *EEDA_CLIENT_EMAIL* configuration options.
- Finally if none of the above method succeeds, the code will check if the current machine is a Google Compute Engine instance, and if so will use the permissions associated to it (using the default service account associated with the VM). To force a machine to be detected as a GCE instance you can set *CPL_MACHINE_IS_GCE* to YES.

7.16.5 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). The following configuration options are available:

- **EEDA_BEARER=value:** Authentication Bearer value to pass to the API. This option is only useful when the token is computed by external code. The bearer validity is typically one hour from the time where it as been requested.
- **EEDA_BEARER_FILE=<filename>:** Similar to *EEDA_BEARER* option, except than instead of passing the value directly, it is the filename where the value should be read.
- **GOOGLE_APPLICATION_CREDENTIALS=<file.json>:** Service account private key file that contains a private key and client email
- **EEDA_PRIVATE_KEY=value:** RSA private key encoded as a PKCS#8 PEM file, with its header and footer. Used together with *EEDA_CLIENT_EMAIL* to use OAuth2 Service Account authentication. Requires GDAL to be built against libcrypto++ or libssl.
- **EEDA_PRIVATE_KEY_FILE=<filename>:** Similar to *EEDA_PRIVATE_KEY* option, except than instead of passing the value directly, it is the filename where the key should be read.
- **EEDA_CLIENT_EMAIL=value:** email to be specified together with *EEDA_PRIVATE_KEY*/*EEDA_PRIVATE_KEY_FILE* to use OAuth2 Service Account authentication.
- **EEDA_PAGE_SIZE=value:** Defaults to `1000`. Features are retrieved from the server by chunks of 1000 by default (and this is the maximum value accepted by the server). This number can be altered with this configuration option.

7.16.6 Attributes

The layer field definition is built by requesting a single image from the collection and guessing the schema from its "properties" element. The "eedaconf.json" file from the GDAL configuration will also be read to check if the collection schema is described in it, in which case the above mentioned schema guessing will not done.

The following attributes will always be present:

Field name	Type	Meaning	Server-side filter compatible
name	String	Image name (e.g. projects/earthengine-public/assets/COPERNICUS/S2/20170430T190351_20170430T190351_T10SEG)	No
id	String	Image ID; equivalent to name without the "projects/*/assets/" prefix (e.g. users/USER/ASSET)	No
path	String	(Deprecated) Image path; equivalent to id	No
gdal_dataset	String	GDAL dataset name (e.g. EEDAI:projects/earthengine-public/assets/COPERNICUS/S2/20170430T190351_20170430T190351_T10SEG) that can be opened with the *EEDAI - Google Earth Engine Data API Image* driver	No
startTime	DateTime	Acquisition start date	**Yes** (restricted to >= comparison on top level)
endTime	DateTime	Acquisition end date	**Yes** (restricted to <= comparison on top level)
updateTime	DateTime	Update date	No
sizeBytes	Integer64	File size in bytes	No
band_count	Integer	Number of bands	No
band_max_width	Integer	Maximum width among bands	No
band_max_height	Integer	Maximum height among bands	No
band_min_pixel_size	Real	Minimum pixel size among bands	No
band_upper_left_x	Real	X origin (only set if equal among all bands)	No
band_upper_left_y	Real	Y origin (only set if equal among all bands)	No
band_crs	String	CRS as EPSG:XXXX or WKT (only set if equal among all bands)	No
other_properties	String	Serialized JSON dictionary with key/value pairs where key is not a standalone field	No

"Server-side filter compatible" means that when this field is included in an attribute filter, it is forwarded to the server (otherwise only client-side filtering is done).

7.16.6.1 Geometry

The footprint of each image is reported as a MultiPolygon with a longitude/latitude WGS84 coordinate system (EPSG:4326).

7.16.6.2 Filtering

The driver will forward any spatial filter set with SetSpatialFilter() to the server. It also makes the same for simple attribute filters set with SetAttributeFilter(). The 3 boolean operators (AND, OR, NOT) and the comparison operators (=, <>, <, <=, > and >=) are supported.

7.16.6.3 Paging

Features are retrieved from the server by chunks of 1000 by default (and this is the maximum value accepted by the server). This number can be altered with the *EEDA_PAGE_SIZE* configuration option.

7.16.6.4 Extent and feature count

The reported extent and feature count will always be respectively (-180,-90,180,90) and -1, given there is no way to get efficient answer to those queries from the server.

7.16.6.5 Examples

Example 1: Listing all images available

```
ogrinfo -ro -al "EEDA:" -oo COLLECTION=projects/earthengine-public/assets/COPERNICUS/S2 -
↪-config EEDA_CLIENT_EMAIL "my@email" --config EEDA_PRIVATE_KEY_FILE my.pem
```

or

```
ogrinfo -ro -al "EEDA:projects/earthengine-public/assets/COPERNICUS/S2" --config EEDA_
↪CLIENT_EMAIL "my@email" --config EEDA_PRIVATE_KEY_FILE my.pem
```

Example 2: Listing all images under a point of (lat,lon)=(40,-100)

```
ogrinfo -ro -al "EEDA:projects/earthengine-public/assets/COPERNICUS/S2" -spat -100 40 -
↪100 40 --config EEDA_CLIENT_EMAIL "my@email" --config EEDA_PRIVATE_KEY_FILE my.pem
```

Example 3: Listing all images available matching criteria

```
ogrinfo -ro -al "EEDA:projects/earthengine-public/assets/COPERNICUS/S2" -where
↪"startTime >= '2015/03/26 00:00:00' AND endTime <= '2015/06/30 00:00:00' AND CLOUDY_
↪PIXEL_PERCENTAGE < 10" --config EEDA_CLIENT_EMAIL "my@email" --config EEDA_PRIVATE_KEY_
↪FILE my.pem
```

7.16.7 See Also:

- *Google Earth Engine Data API Image driver*

7.17 EDIGEO

Driver short name

EDIGEO

Driver built-in by default

This driver is built-in by default

This driver reads files encoded in the French EDIGEO exchange format, a text based file format aimed at exchanging geographical information between GIS, with powerful description capabilities, topology modeling, etc.

The driver has been developed to read files of the French PCI (Plan Cadastral Informatisé - Digital Cadastral Plan) as produced by the DGI (Direction Générale des Impots - General Tax Office). The driver should also be able to open other EDIGEO based products.

The driver must be provided with the .THF file describing the EDIGEO exchange and it will read the associated .DIC, .GEO, .SCD, .QAL and .VEC files.

In order the SRS of the layers to be correctly built, the IGNF file that contains the definition of IGN SRS must be placed in the directory of PROJ resource files.

The whole set of files will be parsed into memory. This may be a limitation if dealing with big EDIGEO exchanges.

7.17.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.17.2 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The following configuration options are available:

- **OGR_EDIGEO_FONT_SIZE_FACTOR=value:** Defaults to `2`. See *Labels*.
- **OGR_EDIGEO_CREATE_LABEL_LAYERS=[YES/NO]:** Defaults to `YES`. See *Labels*.

7.17.3 Labels

For EDIGEO PCI files, the labels are contained in the ID_S_OBJ_Z_1_2_2 layer. OGR will export styling following the *Feature Style Specification*.

It will also add the following fields :

- OGR_OBJ_LNK: the id of the object that is linked to this label

- OBJ_OBJ_LNK_LAYER: the name of the layer of the linked object
- OGR_ATR_VAL: the value of the attribute to display (found in the ATR attribute of the OGR_OBJ_LNK object)
- OGR_ANGLE: the rotation angle in degrees (0 = horizontal, counter-clock-wise oriented)
- OGR_FONT_SIZE: the value of the HEI attribute multiplied by the value of the configuration option *OGR_EDIGEO_FONT_SIZE_FACTOR*.

Combined with the FON (font family) attributes, they can be used to define the styling in QGIS for example.

By default, OGR will create specific layers (xxx_LABEL) to dispatch into the various labels of the ID_S_OBJ_Z_1_2_2 layer according to the value of xxx=OBJ_OBJ_LNK_LAYER. This can be disabled by setting *OGR_EDIGEO_CREATE_LABEL_LAYERS* to NO.

7.17.4 See Also

- Introduction to the EDIGEO standard (in French)
- EDIGEO standard - AFNOR NF Z 52000 (in French)
- Standard d'échange des objets du PCI selon la norme EDIGEO (in French)
- Homepage of the French Digital Cadastral Plan (in French)
- Geotools EDIGEO module description (in English), unmaintained and removed
- Sample of EDIGEO data

7.18 Elasticsearch: Geographically Encoded Objects for Elasticsearch

Driver short name

Elasticsearch

Build dependencies

libcurl

Elasticsearch is an Enterprise-level search engine for a variety of data sources. It supports full-text indexing and geospatial querying of those data in a fast and efficient manor using a predefined REST API.

The driver is known to support ElasticSearch 5.

7.18.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

7.18.2 Opening dataset name syntax

The driver supports reading existing indices from a Elasticsearch host. There are two main possible syntaxes to open a dataset:

- Using *ES:http://hostname:port* (where port is typically 9200)
- Using *ES:* with the open options to specify HOST and PORT

7.18.3 Layer open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are supported:

- **HOST=<hostname>:** Defaults to `localhost`. Server hostname.
- **PORT=<port>:** Defaults to `9200`. Server port.
- **USERPWD=<user:password>:** Basic authentication as username:password.
- **LAYER=<name>:** Index name or index_mapping to use for restricting layer listing.
- **BATCH_SIZE=<integer>:** Defaults to `100`. Number of features to retrieve per batch.
- **FEATURE_COUNT_TO_ESTABLISH_FEATURE_DEFN=<integer>:** Defaults to `100`. Number of features to retrieve to establish feature definition. -1 = unlimited.
- **SINGLE_QUERY_TIMEOUT=<seconds>:** (GDAL >= 3.2.1) Defaults to `unlimited`. Timeout in second (as floating point number) for requests such as GetFeatureCount() or GetExtent().
- **SINGLE_QUERY_TERMINATE_AFTER=<integer>:** (GDAL >= 3.2.1) Defaults to `unlimited`. Maximum number of documents to collect for requests such as GetFeatureCount() or GetExtent().
- **FEATURE_ITERATION_TIMEOUT=<seconds>:** (GDAL >= 3.2.1) Defaults to `unlimited`. Timeout in seconds (as floating point number) for feature iteration, starting from the time of ResetReading().
- **FEATURE_ITERATION_TERMINATE_AFTER=<integer>:** (GDAL >= 3.2.1) Defaults to `unlimited`. Maximum number of documents to collect for feature iteration.
- **JSON_FIELD=[YES/NO]:** Defaults to `NO`. Whether to include a field called "_json" with the full document as JSON.
- **FLATTEN_NESTED_ATTRIBUTE=[YES/NO]:** Defaults to `YES`. Whether to recursively explore nested objects and produce flatten OGR attributes.
- **FID=value:** Defaults to `ogc_fid`. Field name, with integer values, to use as FID.
- **FORWARD_HTTP_HEADERS_FROM_ENV=value:** (GDAL >= 3.1) Can be used to specify HTTP headers, typically for authentication purposes, that must be passed to Elasticsearch. The value of string is a comma separated list of http_header_name=env_variable_name, where http_header_name is the name of a HTTP header and env_variable_name the name of the environment variable / configuration option from which the value of the HTTP header should be retrieved. This is intended for a use case where the OGR Elasticsearch driver is invoked from a web server that stores the HTTP headers of incoming request into environment variables. The ES_FORWARD_HTTP_HEADERS_FROM_ENV configuration option can also be used.
- **AGGREGATION=value:** (GDAL >= 3.5) JSON-serialized definition of an *aggregation*.

7.18.4 Elasticsearch vs OGR concepts

Each mapping type inside a Elasticsearch index will be considered as a OGR layer. A Elasticsearch document is considered as a OGR feature.

7.18.5 Field definitions

Fields are dynamically mapped from the input OGR data source. However, the driver will take advantage of advanced options within Elasticsearch as defined in a field mapping file.

The mapping file allows you to modify the mapping according to the Elasticsearch field-specific types. There are many options to choose from, however, most of the functionality is based on all the different things you are able to do with text fields within Elasticsearch.

```
ogr2ogr -progress --config ES_WRITEMAP /path/to/file/map.txt -f "Elasticsearch" http://
↪localhost:9200 my_shapefile.shp
```

7.18.6 Geometry types

geo_shape is used to store all geometry types (except curve geometries that are not handled by Elasticsearch and will be approximated to their linear equivalents).

7.18.7 Filtering

The driver will forward any spatial filter set with SetSpatialFilter() to the server.

SQL attribute filters set with SetAttributeFilter() are converted to Elasticsearch filter syntax. They will be combined with the potentially defined spatial filter.

It is also possible to directly use a Elasticsearch filter by setting the string passed to SetAttributeFilter() as a JSON serialized object, e.g.

```
{ "post_filter": { "term": { "properties.EAS_ID": 169 } } }
```

Note: if defining directly an Elastic Search JSON filter, the spatial filter specified through SetSpatialFilter() will be ignored, and must thus be included in the JSON filter if needed.

7.18.8 Paging

Features are retrieved from the server by chunks of 100. This can be altered with the BATCH_SIZE open option.

7.18.9 Schema

When reading a Elastic Search index/type, OGR must establish the schema of attribute and geometry fields, since OGR has a fixed schema concept.

In the general case, OGR will read the mapping definition and the first 100 documents (can be altered with the *FEATURE_COUNT_TO_ESTABLISH_FEATURE_DEFN* open option) of the index/type and build the schema that best fit to the found fields and values.

It is also possible to set the *JSON_FIELD=YES* open option so that a _json special field is added to the OGR schema. When reading Elastic Search documents as OGR features, the full JSON version of the document will be stored in the _json field. This might be useful in case of complex documents or with data types that do not translate well in OGR data types. On creation/update of documents, if the _json field is present and set, its content will be used directly (other fields will be ignored).

7.18.10 Feature ID

Elastic Search have a special _id field that contains the unique ID of the document. This field is returned as an OGR field, but cannot be used as the OGR special FeatureID field, which must be of integer type. By default, OGR will try to read a potential 'ogc_fid' field to set the OGR FeatureID. The name of this field to look up can be set with the FID open option. If the field is not found, the FID returned by OGR will be a sequential number starting at 1, but it is not guaranteed to be stable at all.

7.18.11 ExecuteSQL() interface

SQL requests, involving a single layer, with WHERE and ORDER BY statements will be translated as Elasticsearch queries.

Otherwise, if specifying "ES" as the dialect of ExecuteSQL(), a JSON string with a serialized Elastic Search filter can be passed. The search will be done on all indices and types, unless the filter itself restricts the search. The returned layer will be a union of the types returned by the *FEATURE_COUNT_TO_ESTABLISH_FEATURE_DEFN* first documents. It will also contain the _index and _type special fields to indicate the provenance of the features.

The following filter can be used to restrict the search to the "poly" index and its "FeatureCollection" type mapping (Elasticsearch 1.X and 2.X)

```
{ "filter": {
    "indices" : {
        "no_match_filter": "none",
        "index": "poly",
        "filter": {
           "and" : [
             { "type": { "value": "FeatureCollection" } },
             { "term" : { "properties.EAS_ID" : 158.0 } }
           ]
        }
      }
    }
}
```

For Elasticsearch 5.X (works also with 2.X) :

```
{ "post_filter": {
    "indices" : {
        "no_match_query": "none",
        "index": "poly",
        "query": {
          "bool": {
            "must" : [
              { "type": { "value": "FeatureCollection" } },
              { "term" : { "properties.EAS_ID" : 158.0 } }
            ]
          }
        }
      }
    }
}
```

Aggregations are not supported through the ExecuteSQL() interface, but through the below described mechanism.

7.18.12 Aggregations

Added in version 3.5.0.

The driver can support issuing aggregation requests to an index. ElasticSearch aggregations can potentially be rather complex, so the driver currently limits to geohash grid based spatial aggregation, with additional fields with statistical indicators (min, max, average, .), which can be used for example to generate heatmaps. The specification of the aggregation is done through the *AGGREGATION* open option, whose value is a JSON serialized object whose members are:

- `index` (required): the name of the index to query.
- `geometry_field` (optional): the path to the geometry field on which to do geohash grid aggregation. For documents with points encoded as GeoJSON, this will be for example *geometry.coordinates*. When this property is not specified, the driver will analyze the mapping and use the geometry field definition found into it (provided there is a single one). Note that aggregation on geo_shape geometries is only supported since Elasticsearch 7 and may require a non-free license.
- `geohash_grid` (optional): a JSON object, describing a few characteristics of the geohash_grid, that can have the following members:
  - `size` (optional): maximum number of geohash buckets to return per query. The default is 10,000. If `precision` is specified and the number of results would exceed `size`, then the server will trim the results, by sorting by decreasing number of documents matched.
  - `precision` (optional): string length of the geohashes used to define cells/buckets in the results, in the [1,12] range. A geohash of size 1 can generate up to 32 buckets, of size 2 up to 32*32 buckets, etc. When it is not specified, the driver will automatically compute a value, taking into account the `size` parameter and the spatial filter, so that the theoretical number of buckets returned does not exceed `size`.
- `fields` (optional): a JSON object, describing which additional statistical fields should be added, that can have the following members:
  - `min` (optional): array with the paths to index properties on which to compute the minimum during aggregation.
  - `max` (optional): array with the paths to index properties on which to compute the maximum during aggregation.
  - `avg` (optional): array with the paths to index properties on which to compute the average during aggregation.
  - `sum` (optional): array with the paths to index properties on which to compute the sum during aggregation.
  - `count` (optional): array with the paths to index properties on which to compute the value_count during aggregation.
  - `stats` (optional): array with the paths to index properties on which to compute all the above indicators during aggregation.

  When using a GeoJSON mapping, the path to an index property is typically `property.some_name`.

When specifying the *AGGREGATION* open option, a single read-only layer called `aggregation` will be returned. A spatial filter can be set on it using the standard OGR SetSpatialFilter() API: it is applied prior to aggregation.

An example of a potential value for the *AGGREGATION* open option can be:

```
{
    "index": "my_points",
    "geometry_field": "geometry.coordinates",
    "geohash_grid": {
        "size": 1000,
        "precision": 3
```

(continues on next page)

(continued from previous page)

```
    },
    "fields": {
        "min": [ "field_a", "field_b"],
        "stats": [ "field_c" ]
    }
}
```

It will return a layer with a Point geometry field and the following fields:

- `key` of type String: the value of the geohash of the corresponding bucket
- `doc_count` of type Integer64: the number of matching documents in the bucket
- `field_a_min` of type Real
- `field_b_min` of type Real
- `field_c_min` of type Real
- `field_c_max` of type Real
- `field_c_avg` of type Real
- `field_c_sum` of type Real
- `field_c_count` of type Integer64

7.18.13 Multi-target layers

Added in version 3.5.0.

The GetLayerByName() method accepts a layer name that can be a comma-separated list of indices, potentially combined with the '*' wildcard character. See https://www.elastic.co/guide/en/elasticsearch/reference/current/multi-index.html. Note that in the current implementation, the field definition will be established from the one of the matching layers, but not all, so using this functionality will be appropriate when the multiple matching layers share the same schema.

7.18.14 Getting metadata

Getting feature count is efficient.

Getting extent is efficient, only on geometry columns mapped to Elasticsearch type geo_point. On geo_shape fields, feature retrieval of the whole layer is done, which might be slow.

7.18.15 Write support

Index/type creation and deletion is possible.

Write support is only enabled when the datasource is opened in update mode.

When inserting a new feature with CreateFeature() in non-bulk mode, and if the command is successful, OGR will fetch the returned _id and use it for the SetFeature() operation.

7.18.16 Spatial reference system

Geometries stored in Elastic Search are supposed to be referenced as longitude/latitude over WGS84 datum (EPSG:4326). On creation, the driver will automatically reproject from the layer (or geometry field) SRS to EPSG:4326, provided that the input SRS is set and that is not already EPSG:4326.

7.18.17 Layer creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALDatasetCreateLayer()* (C) or *Dataset.CreateLayer* (Python). The driver supports the following layer creation options:

- **INDEX_NAME=value:** Name of the index to create (or reuse). By default the index name is the layer name.
- **INDEX_DEFINITION=[<filename>/<json>]:** Filename from which to read a user-defined index definition, or inlined index definition as serialized JSON .
- **MAPPING_NAME=value:** (Elasticsearch < 7) Name of the mapping type within the index. By default, the mapping name is "FeatureCollection" and the documents will be written as GeoJSON Feature objects. If another mapping name is chosen, a more "flat" structure will be used. This option is ignored when converting to Elasticsearch >=7 (see Removal of mapping types). With Elasticsearch 7 or later, a "flat" structure is always used.
- **MAPPING=[<filename>/<json>]:** Filename from which to read a user-defined mapping, or mapping as serialized JSON .
- **WRITE_MAPPING=<filename>:** Creates a mapping file that can be modified by the user prior to insert in to the index. No feature will be written. This option is exclusive with *MAPPING*.
- **OVERWRITE=[YES/NO]:** Defaults to `NO`. Whether to overwrite an existing type mapping with the layer name to be created.
- **OVERWRITE_INDEX=[YES/NO]:** Defaults to `NO`. Whether to overwrite the whole index to which the layer belongs to. This option is stronger than *OVERWRITE*. *OVERWRITE* will only proceed if the type mapping corresponding to the layer is the single type mapping of the index. In case there are several type mappings, the whole index need to be destroyed (it is unsafe to destroy a mapping and the documents that use it, since they might be used by other mappings. This was possible in Elasticsearch 1.X, but no longer in later versions).
- **GEOMETRY_NAME=value:** Defaults to `geometry`. Name of geometry column.
- **GEOM_MAPPING_TYPE=[AUTO/GEO_POINT/GEO_SHAPE]:** Defaults to `AUTO`. Mapping type for geometry fields. GEO_POINT uses the geo_point mapping type. If used, the "centroid" of the geometry is used. GEO_SHAPE uses the geo_shape mapping type, compatible with all geometry types. When using AUTO, for geometry fields of type Point, a geo_point is used. In other cases, geo_shape is used.
- **GEO_SHAPE_ENCODING=[<GeoJSON>/<WKT>]:** (GDAL >= 3.2.1) Encoding for geo_shape geometry fields. Defaults to GeoJSON. WKT is possible since Elasticsearch 6.2
- **GEOM_PRECISION=<value><unit>:** Desired geometry precision. Number followed by unit. For example 1m. For a geo_point geometry field, this causes a compressed geometry format to be used. This option is without effect if *MAPPING* is specified.
- **STORE_FIELDS=[YES/NO]:** Defaults to `NO`. Whether fields should be stored in the index. Setting to YES sets the "store" property of the field mapping to "true" for all fields. This option is without effect if *MAPPING* is specified.
- **STORED_FIELDS=value:** List of comma separated field names that should be stored in the index. Those fields will have their "store" property of the field mapping set to "true". If all fields must be stored, then using STORE_FIELDS=YES is a shortcut. This option is without effect if *MAPPING* is specified.
- **NOT_ANALYZED_FIELDS=value:** List of comma separated field names that should not be analyzed during indexing. Those fields will have their "index" property of the field mapping set to "not_analyzed" (the default in Elasticsearch is "analyzed"). A same field should not be specified both in *NOT_ANALYZED_FIELDS* and :*NOT_INDEXED_FIELDS*. The {ALL} value can be used to designate all fields. This option is without effect if *MAPPING* is specified.
- **NOT_INDEXED_FIELDS=value:** List of comma separated field names that should not be indexed. Those fields will have their "index" property of the field mapping set to "no" (the default in Elasticsearch is "analyzed").

A same field should not be specified both in *NOT_ANALYZED_FIELDS* and *NOT_INDEXED_FIELDS*. This option is without effect if *MAPPING* is specified.

- **FIELDS_WITH_RAW_VALUE=value:** List of comma separated field names (of type string) that should be created with an additional raw/not_analyzed sub-field, or {ALL} to designate all string analyzed fields. This is needed for sorting on those columns, and can improve performance when filtering with SQL operators. This option is without effect if *MAPPING* is specified.
- **BULK_INSERT=[YES/NO]:** Defaults to `YES`. Whether to use bulk insert for feature creation.
- **BULK_SIZE=<bytes>:** Defaults to `1000000`. Size in bytes of the buffer for bulk upload.
- **FID=value:** Defaults to `ogc_fid`. Field name, with integer values, to use as FID. Can be set to empty to disable the writing of the FID value.
- **DOT_AS_NESTED_FIELD=[YES/NO]:** Defaults to `YES`. Whether to consider dot character in field name as sub-document.
- **IGNORE_SOURCE_ID=[YES/NO]:** Defaults to `NO`. Whether to ignore _id field in features passed to CreateFeature().

7.18.18 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). The following (deprecated) configuration options are available. Layer creation options are also available and should be preferred (see above):

- **ES_WRITEMAP=<filename>:** Creates a mapping file that can be modified by the user prior to insert in to the index. No feature will be written. Note that this will properly work only if only one single layer is created. The *WRITE_MAPPING* layer creation option should be used instead.
- **ES_META=<filename>:** Tells the driver to the user-defined field mappings. The lco:*MAPPING* layer creation option should be used instead.
- **ES_BULK=<bytes>:** Defaults to `5000000`. Identifies the maximum size in bytes of the buffer to store documents to be inserted at a time. Lower record counts help with memory consumption within Elasticsearch but take longer to insert. The *BULK_SIZE* layer creation option should be used instead.
- **ES_OVERWRITE=[YES/NO]:** Defaults to `NO`. Overwrites the current index by deleting an existing one. The *OVERWRITE* layer creation option should be used instead.

7.18.19 Examples

7.18.19.1 Example 1: Open the local store

```
ogrinfo ES:
```

7.18.19.2 Example 2: Open a remote store

```
ogrinfo ES:http://example.com:9200
```

7.18.19.3 Example 3: Filtering on a Elastic Search field

```
ogrinfo -ro ES: my_type -where '{ "post_filter": { "term": { "properties.EAS_ID": 168 } }
↪ }'
```

Using "match" query on Windows: On Windows the query must be between double quotes and double quotes inside the query must be escaped.

```
C:\GDAL_on_Windows>ogrinfo ES: my_type -where "{\"query\": { \"match\": { \"properties.
↪NAME\": \"Helsinki\" } } }"
```

7.18.19.4 Example 4: Basic aggregation

```
ogrinfo -ro ES: my_type -oo "AGGREGATION={\"index\":\"my_points\"}"
```

7.18.19.5 Example 5: Load an Elasticsearch index with a shapefile

```
ogr2ogr -f "Elasticsearch" http://localhost:9200 my_shapefile.shp
```

7.18.19.6 Example 6: Create a Mapping File

The mapping file allows you to modify the mapping according to the Elasticsearch field-specific types. There are many options to choose from, however, most of the functionality is based on all the different things you are able to do with text fields.

```
ogr2ogr -progress -lco WRITE_MAPPING=/path/to/file/map.txt -f "Elasticsearch" http://
↪localhost:9200 my_shapefile.shp
```

7.18.19.7 Example 7: Read the Mapping File

Reads the mapping file during the transformation

```
ogr2ogr -progress -lco MAPPING=/path/to/file/map.txt -f "Elasticsearch" http://
↪localhost:9200 my_shapefile.shp
```

7.18.19.8 Example 8: Bulk Uploading (for larger datasets)

Bulk loading helps when uploading a lot of data. The integer value is the number of bytes that are collected before being inserted. Bulk size considerations

```
ogr2ogr -progress -lco BULK_SIZE=5000000 -f "Elasticsearch" http://localhost:9200 my_
↪shapefile.shp
```

7.18.19.9 Example 9: Overwrite the current Index

If specified, this will overwrite the current index. Otherwise, the data will be appended.

```
ogr2ogr -progress -overwrite ES:http://localhost:9200 PG:"host=localhost user=postgres␣
↪dbname=my_db password=password" "my_table" -nln thetable
```

7.18.20 See Also

- Home page for Elasticsearch
- Examples Wiki
- Google Group

7.19 ESRIJSON / FeatureService driver

Driver short name

ESRIJSON

Driver built-in by default

This driver is built-in by default

This driver can read the JSON output of Feature Service requests following the GeoServices REST Specification, like implemented by ArcGIS Server REST API. The driver can scroll through such result sets that are spread over multiple pages (for ArcGIS servers >= 10.3). This is automatically enabled if URL does not contain an explicit *resultOffset* parameter. If it contains this parameter and scrolling is still desired, the *FEATURE_SERVER_PAGING* open option must be set to YES. The page size can be explicitly set with the *resultRecordCount* parameter (but is subject to a server limit). If it is not set, OGR will set it to the maximum value allowed by the server.

Note: for paged requests to work properly, it is generally necessary to add a sort clause on a field, typically the OBJECTID with a "&orderByFields=OBJECTID+ASC" parameter in the URL, so that the server returns the results in a reliable way.

7.19.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.19.2 Datasource

The driver accepts three types of sources of data:

- Uniform Resource Locator (URL) - a Web address to perform HTTP request.
- Plain text file with ESRIJSON data - identified from the file extension .json
- Text passed directly and encoded in ESRI JSON

The URL/filename/text might be prefixed with ESRIJSON: to avoid any ambiguity with other drivers. Alternatively, starting with GDAL 3.10, specifying the `-if ESRIJSON` option to command line utilities accepting it, or `ESRIJSON` as the only value of the `papszAllowedDrivers` of *GDALOpenEx()*, also forces the driver to recognize the passed URL/filename/text.

7.19.3 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are supported:

- **FEATURE_SERVER_PAGING=[YES/NO]:** Whether to automatically scroll through results with a ArcGIS Feature Service endpoint. Has only effect for ArcGIS servers >= 10.3 and layers with supportsPagination=true capability.
- **HTTP_METHOD=[AUTO/GET/POST]:** (GDAL >= 3.13) Defaults to `AUTO`. Which HTTP request method to use to send requests to the server. In AUTO mode (the default), GET will be used, unless the URL exceeds 256 characters. When it does, the query parameters (the part of the URL after '?') will be sent in the body of a POST request.

7.19.4 Examples

7.19.4.1 Example 1: Read the result of a FeatureService request against a GeoServices REST server

Note that this server does not support paging.

```
ogrinfo -ro -al "https://sampleserver6.arcgisonline.com/arcgis/rest/services/PoolPermits/
↪FeatureServer/0/query?resultRecordCount=10&f=pjson"
```

7.19.5 See Also

- *GeoJSON driver*
- GeoServices REST Specification

7.20 ESRI File Geodatabase (FileGDB)

Driver short name

FileGDB

Build dependencies

FileGDB API library

The FileGDB driver provides read and write access to vector layers of File Geodatabases (.gdb directories) created by ArcGIS 10 and above. The dataset name must be the directory/folder name, and it must end with the .gdb extension.

Note : the *OpenFileGDB driver* driver exists as an alternative built-in (i.e. not depending on a third-party library) driver.

Starting with GDAL 3.11, update and creation support is delegated to the *OpenFileGDB driver* driver.

7.20.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

7.20.2 Requirements

ESRI File Geodatabase API

Curve in geometries are supported on reading.

7.20.3 SQL support

SQL statements are run through the SQL engine of the FileGDB SDK API. This holds for non-SELECT statements. However, due to partial/inaccurate support for SELECT statements in current FileGDB SDK API versions (v1.2), SELECT statements will be run by default by the OGR SQL engine. This can be changed by specifying the *-dialect FileGDB* option to ogrinfo or ogr2ogr.

7.20.3.1 Special SQL requests

"GetLayerDefinition a_layer_name" and "GetLayerMetadata a_layer_name" can be used as special SQL requests to get respectively the definition and metadata of a FileGDB table as XML content.

Starting with GDAL 3.5, the "REPACK" special SQL request can be issued to ask for database compaction.

7.20.4 Field domains

Added in version 3.3.

Retrieving coded and range field domains are supported. Writing support has been added in GDAL 3.5.

7.20.5 Relationships

Added in version 3.6.

Relationship retrieval is supported.

7.20.6 Hierarchical organization

Added in version 3.4.

The hierarchical organization of tables and feature classes as top-level element or within a feature dataset can be explored using the methods *GDALDataset::GetRootGroup()*, *GDALGroup::GetGroupNames()*, *GDALGroup::OpenGroup()*, *GDALGroup::GetVectorLayerNames()* and *GDALGroup::OpenVectorLayer()*

7.20.7 Geometry coordinate precision

Added in version GDAL: 3.9

The driver supports reading and writing the geometry coordinate precision, using the XYResolution, ZResolution and MResolution members of the *OGRGeomCoordinatePrecision* settings of the *OGRGeomFieldDefn*. `XYScale` is computed as 1.0 / `XYResolution` (and similarly for the Z and M components). The tolerance setting is computed as being one tenth of the resolution

On reading, the coordinate precision grid parameters are returned as format specific options of *OGRGeomCoordinatePrecision* with the `FileGeodatabase` format key, with the following option key names: `XYScale`, `XYTolerance`, `XYOrigin`, `ZScale`, `ZTolerance`, `ZOrigin`, `MScale`, `MTolerance`, `MOrigin`. On writing, they are also honored (they will have precedence over XYResolution, ZResolution and MResolution).

On layer creation, the XORIGIN, YORIGIN, ZORIGIN, MORIGIN, XYSCALE, ZSCALE, ZORIGIN, XYTOLERANCE, ZTOLERANCE, MTOLERANCE layer creation options will be used in priority over the settings of *OGRGeomCoordinatePrecision*.

7.20.8 Limitations

- The SDK is known to be unable to open layers with particular spatial reference systems. This might be the case if messages "FGDB: Error opening XXXXXXX. Skipping it (Invalid function arguments.)" when running `ogrinfo --debug on the.gdb` (reported as warning in GDAL 2.0). Using the OpenFileGDB driver will generally solve that issue.
- FGDB coordinate snapping will cause geometries to be altered during writing. Use the origin and scale layer creation options to control the snapping behavior.
- Reading data compressed in SDC format (Smart Data Compression) is not support by the driver, because it is not supported by the ESRI SDK.
- Reading data compressed in CDF format (Compressed Data Format) requires ESRI SDK 1.4 or later.
- Some applications create FileGeodatabases with non-spatial tables which are not present in the GDB_Items metadata table. These tables cannot be opened by the ESRI SDK, so GDAL will automatically fallback to the OpenFileGDB driver to read these tables. Accordingly they will be opened with the limitations of the OpenFileGDB driver (for instance, they will be read only).
- The driver does not support 64-bit integers.

7.20.9 Links

- ESRI File Geodatabase API Page
- *OpenFileGDB driver*, not depending on a third-party library/SDK

7.21 FlatGeobuf

Added in version 3.1.

Driver short name

FlatGeobuf

Driver built-in by default

This driver is built-in by default

This driver implements read/write support for access to features encoded in FlatGeobuf format, a performant binary encoding for geographic data based on flatbuffers that can hold a collection of Simple Features.

7.21.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.21.2 Multi layer support

A single .fgb file only contains one single layer. For multiple layer support, it is possible to put several .fgb files in a directory, and use that directory name as the connection string.

On creation, passing a filename without a .fgb suffix will instruct the driver to create a directory of that name, and create layers as .fgb files in that directory.

Starting with GDAL 3.9, metadata set at the layer level will be written in the FlatGeobuf header, and retrieved on reading as layer metadata.

7.21.3 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are supported:

- **VERIFY_BUFFERS=[YES/NO]:** Defaults to `YES`. Set to YES to verify buffers when reading. This can provide some protection for invalid/corrupt data with a performance trade off.

7.21.4 Dataset Creation Options

None

7.21.5 Layer Creation Options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALDatasetCreateLayer()`* (C) or *`Dataset.CreateLayer`* (Python). The following layer creation options are supported:

- **SPATIAL_INDEX=[YES/NO]:** Defaults to `YES`. Set to YES to create a spatial index.
- **TEMPORARY_DIR=<path>:** Path to an existing directory where temporary files should be created. Only used if *`SPATIAL_INDEX=YES`*. If not specified, the directory of the output file will be used for regular filenames. For other VSI file systems, the temporary directory will be the one decided by the *`CPLGenerateTempFilename()`* function. "/vsimem/" can be used for in-memory temporary files.
- **TITLE=<string>:** (GDAL >= 3.9) Dataset title (should be relatively short)
- **DESCRIPTION=<string>:** (GDAL >= 3.9) Dataset description (intended for free form long text)

7.21.6 Creation Issues

- Currently, NULL geometries are not supported with *`SPATIAL_INDEX=YES`*.

  More background and discussion on this issue at

- The creation of the packet Hilbert R-Tree requires an amount of RAM which is at least the number of features times 83 bytes.

7.21.7 Examples

7.21.7.1 Example 1: Simple translation of a shapefile into a FlatGeobuf file

The file 'filename.fgb' will be created with the features from abc.shp and attributes from abc.dbf. The file `filename.fgb` must **not** already exist, as it will be created.

```
ogr2ogr -f FlatGeobuf filename.fgb abc.shp
```

7.21.7.2 Example 2: Conversion of a Geopackage file with multiple layers

```
ogr2ogr -f FlatGeobuf my_fgb_dataset input.gpkg
```

7.21.8 See Also

- FlatGeobuf at GitHub

7.22 GDALG: GDAL Streamed Algorithm

Added in version 3.11.

Driver short name

GDALG

Driver built-in by default

This driver is built-in by default

Note

GDALG is the contraction of GDAL and ALGorithm.

This is a read-only driver that reads a JSON file containing an invocation of the *gdal command line interface*, that results in a on-the-fly / streamed vector dataset when used as an input to *`GDALOpenEx()`*, or anywhere else in GDAL where a vector input dataset is expected. GDALG files are conceptually close to *VRT (Virtual) files*, although the implementation is substantially different.

The subset of commands of **`gdal`** that can generate such streamed datasets indicate support for writing `GDALG` in the `--help` description for `--output-format`. This includes *gdal vector pipeline*, *gdal vector reproject*, *gdal vector filter*, *gdal vector select*, *gdal vector sql*, etc.

It is recommended that GDALG files use the `.gdalg.json` extension.

The JSON document must include a `type` member with the value `gdal_streamed_alg`, and a `command_line` member, which is very close to the one that would be used to generate a materialized dataset, but using the `stream` output format.

```
{
    "type": "gdal_streamed_alg",
    "command_line": "gdal vector reproject --dst-crs=EPSG:32632 in.gpkg --output-format␣
```

(continues on next page)

(continued from previous page)

```
↪stream streamed_dataset"
}
```

If such file is saved as `reprojected.gdalg.json`, it can be for example read with `gdal vector info reprojected.gdalg.json`.

In the case of a *gdal vector pipeline*, the final `write` step can be omitted.

```
{
    "type": "gdal_streamed_alg",
    "command_line": "gdal vector pipeline ! read in.tif ! reproject --dst-crs=EPSG:32632␣
↪! select --fields fid,geom"
}
```

An optional `relative_paths_relative_to_this_file` boolean member defaults to `true`, to indicate that relative input filenames in the command line should be interpreted as relative to the location of the `.gdalg.json` file. If setting this member to `false`, they will be interpreted as being relative to the current working directory.

`.gdalg.json` files can be validated against the following JSON schema gdalg.schema.json

This driver has also raster capabilities as detailed in *GDALG: GDAL Streamed Algorithm*.

7.22.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

7.23 GeoJSON

Driver short name

GeoJSON

Driver built-in by default

This driver is built-in by default

This driver implements read/write support for access to features encoded in GeoJSON format. GeoJSON is a dialect based on the JavaScript Object Notation (JSON). JSON is a lightweight plain text format for data interchange and GeoJSON is nothing other than its specialization for geographic content.

GeoJSON is supported as an output format of a number of services: GeoServer, CartoWeb, etc.

The OGR GeoJSON driver translates GeoJSON encoded data to objects of the OGR Simple Features model: Datasource, Layer, Feature, Geometry. The implementation is based on GeoJSON Specification, v1.0.

The GeoJSON driver supports updating existing GeoJSON files. In that case, the default value for the NATIVE_DATA open option will be YES.

7.23.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.23.2 Datasource

The OGR GeoJSON driver accepts three types of sources of data:

- Uniform Resource Locator (URL) - a Web address to perform HTTP request
- Plain text file with GeoJSON data - identified from the file extension .geojson or .json
- Text passed directly and encoded in GeoJSON

The URL/filename/text might be prefixed with GeoJSON: to avoid any ambiguity with other drivers. Alternatively, starting with GDAL 3.10, specifying the `-if GeoJSON` option to command line utilities accepting it, or `GeoJSON` as the only value of the `papszAllowedDrivers` of *GDALOpenEx()*, also forces the driver to recognize the passed URL/filename/text.

7.23.3 Layer

A GeoJSON datasource is translated to single OGRLayer object with pre-defined name *OGRGeoJSON*:

```
ogrinfo -ro http://featureserver/data/.geojson OGRGeoJSON
```

It is also valid to assume that OGRDataSource::GetLayerCount() for GeoJSON datasource always returns 1.

The layer name is built with the following logic:

1. If a "name" member is found at the FeatureCollection level, it is used.
2. Otherwise if the filename is regular (ie not a URL with query parameters), then the filename without extension and path is used as the layer name.
3. Otherwise OGRGeoJSON is used.

Accessing Web Service as a datasource (i.e. FeatureServer), each request will produce a new layer. This behavior conforms to the stateless nature of HTTP transaction and is similar to how Web browsers operate: single request == single page.

If a top-level member of GeoJSON data is of any other type than *FeatureCollection*, the driver will produce a layer with only one feature. Otherwise, a layer will consist of a set of features.

If the *NATIVE_DATA* open option is set to YES, members at the level of the FeatureCollection will be stored as a serialized JSON object in the NATIVE_DATA item of the NATIVE_DATA metadata domain of the layer object

(and "application/geo+json" in the NATIVE_MEDIA_TYPE of the NATIVE_DATA metadata domain, or "application/vnd.geo+json" before GDAL 3.13.1).

7.23.4 Feature

The OGR GeoJSON driver maps each object of following types to new *OGRFeature* object: Point, LineString, Polygon, GeometryCollection, Feature.

According to the *GeoJSON Specification*, only the *Feature* object must have a member with name *properties*. Each and every member of *properties* is translated to OGR object of type of OGRField and added to corresponding OGRFeature object.

The *GeoJSON Specification* does not require all *Feature* objects in a collection to have the same schema of properties. If *Feature* objects in a set defined by *FeatureCollection* object have different schema of properties, then resulting schema of fields in OGRFeatureDefn is generated as union of all *Feature* properties.

Schema detection will recognize fields of type String, Integer, Real, StringList, IntegerList and RealList, Integer(Boolean), Date, Time and DateTime.

It is possible to tell the driver not to process attributes by setting the configuration option `ATTRIBUTES_SKIP=YES`. The default behavior is to preserve all attributes (as a union, see previous paragraph), which is equal to setting `ATTRIBUTES_SKIP=NO`.

If the `NATIVE_DATA` open option is set to YES, the Feature JSON object will be stored as a serialized JSON object in the NativeData property of the OGRFeature object (and "application/geo+json" in the NativeMediaType property, or "application/vnd.geo+json" before GDAL 3.13.1). On write, if the OGRFeature to be written has its NativeMediaType property set to "application/geo+json" (or "application/vnd.geo+json" before GDAL 3.13.1) and its NativeData property set to a string that is a serialized JSON object, then extra members of this object (i.e. not the "property" dictionary, nor the first 3 dimensions of geometry coordinates) will be used to enhance the created JSON object from the OGRFeature. See rfc-60 for more details.

7.23.5 Geometry

Similarly to the issue with mixed-properties features, the *GeoJSON Specification* draft does not require that all *Feature* objects in a collection have the same type of geometry. Fortunately, OGR objects model does allow to have geometries of different types in single layer - a heterogeneous layer. By default, the GeoJSON driver preserves type of geometries.

However, sometimes the need arises to have a homogeneous layer from a set of heterogeneous features. For this purpose, it is possible to tell the driver to wrap all geometries with OGRGeometryCollection type as a common denominator. This behavior may be controlled by setting the `GEOMETRY_AS_COLLECTION` configuration option to YES.

7.23.6 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as `CPLSetConfigOption()` (C) or `gdal.config_options` (Python). The following configuration options are available:

- **GEOMETRY_AS_COLLECTION=[YES/NO]:** Defaults to `NO`. used to control translation of geometries: YES: wrap geometries with OGRGeometryCollection type
- **ATTRIBUTES_SKIP=[YES/NO]:** Controls translation of attributes. If `YES`, skip all attributes.
- **OGR_GEOJSON_ARRAY_AS_STRING=value:** Equivalent of `ARRAY_AS_STRING` open option.
- **OGR_GEOJSON_DATE_AS_STRING=value:** Equivalent of `DATE_AS_STRING` open option.
- **OGR_GEOJSON_MAX_OBJ_SIZE=<MBytes>:** (GDAL >= 3.0.2) Defaults to `200`. size in MBytes of the maximum accepted single feature, or 0 to allow for a unlimited size (GDAL >= 3.5.2).

7.23.7 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). This driver supports the following open options:

- **FLATTEN_NESTED_ATTRIBUTES=[YES/NO]:** Defaults to `NO`. Whether to recursively explore nested objects and produce flatten OGR attributes.
- **NESTED_ATTRIBUTE_SEPARATOR=<character>:** Defaults to `_`. Separator between components of nested attributes.
- **FEATURE_SERVER_PAGING=[YES/NO]:** Whether to automatically scroll through results with a ArcGIS Feature Service endpoint.
- **NATIVE_DATA=[YES/NO]:** Defaults to `NO`. Whether to store the native JSON representation at FeatureCollection and Feature level. This option can be used to improve round-tripping from GeoJSON to GeoJSON by preserving some extra JSON objects that would otherwise be ignored by the OGR abstraction. Note that ogr2ogr by default enable this option, unless you specify its -noNativeData switch.
- **ARRAY_AS_STRING=[YES/NO]:** Whether to expose JSON arrays of strings, integers or reals as a OGR String. Default is NO. Can also be set with the *OGR_GEOJSON_ARRAY_AS_STRING* configuration option.
- **DATE_AS_STRING=[YES/NO]:** (GDAL >= 3.0.3) Defaults to `NO`. Whether to expose date/time/date-time content using dedicated OGR date/time/date-time types or as a OGR String. Default is NO (that is date/time/date-time are detected as such). Can also be set with the *OGR_GEOJSON_DATE_AS_STRING* configuration option.
- **FOREIGN_MEMBERS=[AUTO/ALL/NONE/STAC]:** (GDAL >= 3.11.0) Defaults to `AUTO`. Whether and how foreign members at the feature level should be processed as OGR fields:
  - `AUTO` mode behaves like `STAC` mode if a `stac_version` member is found at the Feature level, otherwise it behaves as `NONE` mode.
  - In `ALL` mode, all foreign members at the feature level are added. Whether to recursively explore nested objects and produce flatten OGR attributes or not is decided by the `FLATTEN_NESTED_ATTRIBUTES` open option.
  - In `NONE` mode, no foreign members at the feature level are added.
  - `STAC` mode (Spatio-Temporal Asset Catalog) behaves the same as `ALL`, except content under the `assets` member is by default flattened as `assets.{asset_name}.{asset_property}` fields.
- **OGR_SCHEMA=<filename>|<json string>:** (GDAL >= 3.11.0) Partially or totally overrides the auto-detected schema to use for creating the layer. The overrides are defined as a JSON list of field definitions. This can be a filename, a URL or JSON string conformant with the ogr_fields_override.schema.json schema

To explain *FLATTEN_NESTED_ATTRIBUTES*, consider the following GeoJSON fragment:

```
{
  "type": "FeatureCollection",
  "features":
  [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [ 2, 49 ]
      },
      "properties": {
        "a_property": "foo",
```

(continues on next page)

(continued from previous page)

```
        "some_object": {
          "a_property": 1,
          "another_property": 2
        }
      }
    }
  ]
}
```

"ogrinfo test.json -al -oo FLATTEN_NESTED_ATTRIBUTES=yes" reports:

```
OGRFeature(OGRGeoJSON):0
  a_property (String) = foo
  some_object_a_property (Integer) = 1
  some_object_another_property (Integer) = 2
  POINT (2 49)
```

7.23.8 Layer creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALDatasetCreateLayer()* (C) or *Dataset.CreateLayer* (Python). This driver supports the following layer creation options:

- **WRITE_BBOX=[YES/NO]:** Defaults to `NO`. Set to YES to write a bbox property with the bounding box of the geometries at the feature and feature collection level.
- **COORDINATE_PRECISION=<integer>:** Maximum number of figures after decimal separator to write in coordinates. Default to 15 for GeoJSON 2008, and 7 for RFC 7946. "Smart" truncation will occur to remove trailing zeros.
- **SIGNIFICANT_FIGURES=<integer>:** Defaults to `17`. Maximum number of significant figures when writing floating-point numbers. If explicitly specified, and *COORDINATE_PRECISION* is not, this will also apply to coordinates.
- **NATIVE_DATA=value:** Serialized JSON object that contains extra properties to store at FeatureCollection level.
- **NATIVE_MEDIA_TYPE=value:** Format of *NATIVE_DATA*. Must be "application/geo+json" (or "application/vnd.geo+json" before GDAL 3.13.1), otherwise *NATIVE_DATA* will be ignored.
- **RFC7946=[YES/NO]:** Defaults to `NO`. Whether to use RFC 7946 standard. Otherwise GeoJSON 2008 initial version will be used. Default is NO (thus GeoJSON 2008)
- **WRAPDATELINE=[YES/NO]:** (GDAL >= 3.5.2) Defaults to `YES`. Whether to apply heuristics to split geometries that cross dateline. Only used when coordinate transformation occurs or when *RFC7946=YES*. Default is YES (and also the behavior for OGR < 3.5.2).
- **WRITE_NAME=[YES/NO]:** Defaults to `YES`. Whether to write a "name" property at feature collection level with layer name.
- **DESCRIPTION=value:** (Long) description to write in a "description" property at feature collection level. On reading, this will be reported in the DESCRIPTION metadata item of the layer.
- **ID_FIELD=value:** Name of the source field that must be written as the 'id' member of Feature objects.
- **ID_TYPE=[AUTO/String/Integer]:** Type of the 'id' member of Feature objects.
- **ID_GENERATE=[YES/NO]:** (GDAL >= 3.1) Auto-generate feature ids

- **WRITE_NON_FINITE_VALUES=[YES/NO]:** Defaults to `NO`. Whether to write NaN / Infinity values. Such values are not allowed in strict JSON mode, but some JSON parsers (libjson-c >= 0.12 for example) can understand them as they are allowed by ECMAScript.
- **AUTODETECT_JSON_STRINGS=[YES/NO]:** (GDAL >= 3.8) Defaults to `YES`. Whether to try to interpret string fields as JSON arrays or objects if they start and end with brackets and braces, even if they do not have their subtype set to JSON.
- **FOREIGN_MEMBERS_FEATURE=value:** (GDAL >= 3.9) JSON serialized object whose content must be merged into each Feature object. The string should start with { and end with }. Those characters will be striped off in the output stream. It is the responsibility of the user to ensure that the added foreign members are different from the other members of the Feature, such as "type", "id", "properties", "geometry".
- **FOREIGN_MEMBERS_COLLECTION=value:** (GDAL >= 3.9) JSON serialized object whose content must be merged into the FeatureCollection object. The string should start with { and end with }. Those characters will be striped off in the output stream. It is the responsibility of the user to ensure that the added foreign members are different from the other members of the FeatureCollection, such as "type", "name", "crs", "features".

7.23.9 VSI Virtual File System API support

The driver supports reading and writing to files managed by VSI Virtual File System API, which includes "regular" files, as well as files in the /vsizip/ (read-write), /vsigzip/ (read-write), /vsicurl/ (read-only) domains.

Writing to /dev/stdout or /vsistdout/ is also supported.

7.23.10 Round-tripping of extra JSON members

See rfc-60 for more details.

Extra JSON members at the FeatureCollection, Feature or geometry levels that are not normally reflected in the OGR abstraction, such as the ones called "extra_XXXXX_member" in the below snippet, are by default preserved when executing ogr2ogr with GeoJSON both at the source and destination. This also applies to extra values in position tuples of geometries, beyond the 3rd dimension (100, 101 in the below example), if the transformation preserves the geometry structure (for example, reprojection is allowed, but not change in the number of coordinates).

```
{
  "type": "FeatureCollection",
  "extra_fc_member": "foo",
  "features":
  [
    {
      "type": "Feature",
      "extra_feat_member": "bar",
      "geometry": {
        "type": "Point",
        "extra_geom_member": "baz",
        "coordinates": [ 2, 49, 3, 100, 101 ]
      },
      "properties": {
        "a_property": "foo",
      }
    }
  ]
}
```

This behavior can be turned off by specifying the **-noNativeData** switch of the ogr2ogr utility.

7.23.11 RFC 7946 write support

By default, the driver will write GeoJSON files following GeoJSON 2008 specification. When specifying the *RFC7946=YES* creation option, the RFC 7946 standard will be used instead.

The differences between the 2 versions are mentioned in Appendix B of RFC 7946 and recalled here for what matters to the driver:

- Coordinates must be geographic over the WGS 84 ellipsoid, hence if the spatial reference system specified at layer creation time is not EPSG:4326, on-the-fly reprojection will be done by the driver.
- Polygons will be written such as to follow the right-hand rule for orientation (counterclockwise external rings, clockwise internal rings).
- The values of a "bbox" array are "[west, south, east, north]", not "[minx, miny, maxx, maxy]"
- Some extension member names (see previous section about round/tripping) are forbidden in the FeatureCollection, Feature and Geometry objects.
- The default coordinate precision is 7 decimal digits after decimal separator.

7.23.12 Geometry coordinate precision

Added in version GDAL: 3.9

The GeoJSON driver supports reading and writing the geometry coordinate precision, using the *OGRGeomCoordinatePrecision* settings of the *OGRGeomFieldDefn* Those settings are used to round the coordinates of the geometry of the features to an appropriate decimal precision.

> **Note**
>
> The *COORDINATE_PRECISION* layer creation option has precedence over the values set on the *OGRGeomFieldDefn*.

Implementation details: the coordinate precision is stored as `xy_coordinate_resolution` and `z_coordinate_resolution` members at the FeatureCollection level. Their numeric value is expressed in the units of the SRS.

Example:

```
{
    "type": "FeatureCollection",
    "xy_coordinate_resolution": 8.9e-6,
    "z_coordinate_resolution": 1e-1,
    "features": []
}
```

7.23.13 Examples

7.23.13.1 Example 1: Dumping content of a .geojson file

```
ogrinfo -ro point.geojson
```

7.23.13.2 Example 2: Querying features from remote service with filtering by attribute

```
ogrinfo -ro http://featureserver/cities/.geojson OGRGeoJSON -where "name=Warsaw"
```

7.23.13.3 Example 3: Translating number of features queried from FeatureServer to ESRI Shapefile

```
ogr2ogr -f "ESRI Shapefile" cities.shp http://featureserver/cities/.geojson OGRGeoJSON
```

7.23.13.4 Example 4: Converting a Shapefile into a RFC 7946 GeoJSON file

```
ogr2ogr -f GeoJSON cities.json cities.shp -lco RFC7946=YES
```

7.23.14 See Also

- GeoJSON - encoding geographic content in JSON
- RFC 7946 standard.
- GeoJSON 2008 specification (obsoleted by RFC 7946).
- JSON - JavaScript Object Notation
- *GeoJSON sequence driver*
- *OGC Features and Geometries JSON (JSON-FG) driver*
- *ESRI JSON / FeatureService driver*
- *TopoJSON driver*

7.24 GeoJSONSeq: sequence of GeoJSON features

Driver short name

GeoJSONSeq

Driver built-in by default

This driver is built-in by default

This driver implements read/creation support for features encoded individually as GeoJSON Feature objects, separated by newline (LF) (Newline Delimited JSON) or record-separator (RS) characters (RFC 8142 standard: GeoJSON Text Sequences)

Such files are equivalent to a GeoJSON FeatureCollection, but are more friendly for incremental parsing.

The driver automatically reprojects geometries to WGS84 longitude, latitude, if the layer is created with another SRS.

Appending to an existing file is supported since GDAL 3.6

7.24.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.24.2 Datasource

The driver accepts three types of sources of data:

- Uniform Resource Locator (URL) - a Web address to perform HTTP request
- Plain text file with GeoJSON data - identified from the file extension .geojsonl or .geojsons
- Text passed directly as filename, and encoded as GeoJSON sequences

The URL/filename/text might be prefixed with GeoJSONSeq: to avoid any ambiguity with other drivers. Alternatively, starting with GDAL 3.10, specifying the `-if GeoJSONSeq` option to command line utilities accepting it, or `GeoJSONSeq` as the only value of the `papszAllowedDrivers` of *`GDALOpenEx()`*, also forces the driver to recognize the passed URL/filename/text.

7.24.3 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The following configuration option is available:

- **OGR_GEOJSON_MAX_OBJ_SIZE=<MBytes>:** (GDAL >= 3.0.2) Defaults to `200`. size in MBytes of the maximum accepted single feature, or 0 to allow for a unlimited size (GDAL >= 3.5.2).

7.24.4 Layer creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALDatasetCreateLayer()`* (C) or *`Dataset.CreateLayer`* (Python). The following layer creation options are supported:

- **RS=[YES/NO]:** whether to start records with the RS=0x1E character, so as to be compatible with the RFC 8142 standard. Defaults to NO, unless the filename extension is "geojsons"
- **COORDINATE_PRECISION=<integer>:** Defaults to `7`. Maximum number of figures after decimal separator to write in coordinates. "Smart" truncation will occur to remove trailing zeros.
- **SIGNIFICANT_FIGURES=<integer>:** Defaults to `17`. Maximum number of significant figures when writing floating-point numbers. If explicitly specified, and *`COORDINATE_PRECISION`* is not, this will also apply to coordinates.
- **ID_FIELD=value:** Name of the source field that must be written as the 'id' member of Feature objects.
- **ID_TYPE=[AUTO/String/Integer]:** Type of the 'id' member of Feature objects.
- **WRITE_NON_FINITE_VALUES=[YES/NO]:** (GDAL >= 3.8) Defaults to `NO`. Whether to write NaN / Infinity values. Such values are not allowed in strict JSON mode, but some JSON parsers (libjson-c >= 0.12 for example) can understand them as they are allowed by ECMAScript.

- **AUTODETECT_JSON_STRINGS=[YES/NO]:** (GDAL >= 3.8) Defaults to `YES`. Whether to try to interpret string fields as JSON arrays or objects if they start and end with brackets and braces, even if they do not have their subtype set to JSON.
- **WRITE_BBOX=[YES/NO]:** (GDAL >= 3.10) Defaults to `NO`. Set to YES to write a bbox property with the bounding box of the geometry at the feature level.

7.24.5 Geometry coordinate precision

Added in version GDAL: 3.9

On creation, the GeoJSONSeq driver supports using the geometry coordinate precision, from the *`OGRGeomCoordinatePrecision`* settings of the *`OGRGeomFieldDefn`*. Those settings are used to round the coordinates of the geometry of the features to an appropriate decimal precision.

> **Note**
>
> The *`COORDINATE_PRECISION`* layer creation option has precedence over the values set on the *`OGRGeomFieldDefn`*.

The value of those geometry coordinate precision is *not* serialized in the generated file, hence on reading, the driver will not advertise a geometry coordinate precision.

7.24.6 See Also

- *GeoJSON driver*
- RFC 7946 standard: the GeoJSON Format.
- RFC 8142 standard: GeoJSON Text Sequences (RS separator)
- GeoJSONL: An optimized format for large geographic datasets
- JSON streaming on Wikipedia: An overview over formats for concatenated JSON in a single file

7.25 GeoRSS : Geographically Encoded Objects for RSS feeds

> **Driver short name**
>
> GeoRSS

> **Build dependencies**
>
> (read support needs libexpat)

GeoRSS is a way of encoding location in RSS or Atom feeds.

OGR has support for GeoRSS reading and writing. Read support is only available if GDAL is built with *expat* library support

The driver supports RSS documents in RSS 2.0 or Atom 1.0 format.

It also supports the 3 ways of encoding location : GeoRSS simple, GeoRSS GML and W3C Geo (the later being deprecated).

The driver can read and write documents without location information as well.

The default datum for GeoRSS document is the WGS84 datum (EPSG:4326). Although that GeoRSS locations are encoded in latitude-longitude order in the XML file, all coordinates reported or expected by the driver are in longitude-latitude order. The longitude/latitude order used by OGR is meant for compatibility with most of the rest of OGR drivers and utilities. For locations encoded in GML, the driver will support the srsName attribute for describing other SRS.

Simple and GML encoding support the notion of a *box* as a geometry. This will be decoded as a rectangle (Polygon geometry) in OGR Simple Feature model.

A single layer is returned while reading a RSS document. Features are retrieved from the content of <item> (RSS document) or <entry> (Atom document) elements.

7.25.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.25.2 Encoding issues

Expat library supports reading the following built-in encodings :

- US-ASCII
- UTF-8
- UTF-16
- ISO-8859-1
- Windows-1252

The content returned by OGR will be encoded in UTF-8, after the conversion from the encoding mentioned in the file header is.

If your GeoRSS file is not encoded in one of the previous encodings, it will not be parsed by the GeoRSS driver. You may convert it into one of the supported encoding with the *iconv* utility for example and change accordingly the *encoding* parameter value in the XML header.

When writing a GeoRSS file, the driver expects UTF-8 content to be passed in.

7.25.3 Field definitions

While reading a GeoRSS document, the driver will first make a full scan of the document to get the field definitions.

The driver will return elements found in the base schema of RSS channel or Atom feeds. It will also return extension elements, that are allowed in namespaces.

Attributes of first level elements will be exposed as fields.

Complex content (elements inside first level elements) will be returned as an XML blob.

When a same element is repeated, a number will be appended at the end of the attribute name for the repetitions. This is useful for the <category> element in RSS and Atom documents for example.

The following content :

```
<item>
    <title>My tile</title>
    <link>http://www.mylink.org</link>
    <description>Cool description !</description>
    <pubDate>Wed, 11 Jul 2007 15:39:21 GMT</pubDate>
    <guid>http://www.mylink.org/2007/07/11</guid>
    <category>Computer Science</category>
    <category>Open Source Software</category>
    <georss:point>49 2</georss:point>
    <myns:name type="my_type">My Name</myns:name>
    <myns:complexcontent>
        <myns:subelement>Subelement</myns:subelement>
    </myns:complexcontent>
</item>
```

will be interpreted in the OGR SF model as :

```
title (String) = My title
link (String) = http://www.mylink.org
description (String) = Cool description !
pubDate (DateTime) = 2007/07/11 15:39:21+00
guid (String) = http://www.mylink.org/2007/07/11
category (String) = Computer Science
category2 (String) = Open Source Software
myns_name (String) = My Name
myns_name_type (String) = my_type
myns_complexcontent (String) = <myns:subelement>Subelement</myns:subelement>
POINT (2 49)
```

7.25.4 Creation Issues

On export, all layers are written to a single file. Update of existing files is not supported.

If the output file already exits, the writing will not occur. You have to delete the existing file first.

A layer that is created cannot be immediately read without closing and reopening the file. That is to say that a dataset is read-only or write-only in the same session.

Supported geometries :

- Features of type wkbPoint/wkbPoint25D.
- Features of type wkbLineString/wkbLineString25D.

- Features of type wkbPolygon/wkbPolygon25D.

Other type of geometries are not supported and will be silently ignored.

7.25.5 Dataset creation options

Dataset creation options can be specified in command-line tools using the syntax `-dsco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following dataset creation options are supported:

- **FORMAT=[RSS/ATOM]:** Defaults to `RSS`. whether the document must be in RSS 2.0 or Atom 1.0 format.
- **GEOM_DIALECT=[SIMPLE/GML/W3C_GEO]:** Defaults to `simple`. the encoding of location information. W3C_GEO only supports point geometries. SIMPLE or W3C_GEO only support geometries in geographic WGS84 coordinates.
- **USE_EXTENSIONS=[YES/NO]:** Defaults to `NO`. If defined to YES, extension fields (that is to say fields not in the base schema of RSS or Atom documents) will be written. If the field name not found in the base schema matches the foo_bar pattern, foo will be considered as the namespace of the element, and a <foo:bar> element will be written. Otherwise, elements will be written in the <ogr:> namespace.
- **WRITE_HEADER_AND_FOOTER=[YES/NO]:** Defaults to `YES`. If defined to NO, only <entry> or <item> elements will be written. The user will have to provide the appropriate header and footer of the document. Following options are not relevant in that case.
- **HEADER=value:** (RSS or Atom document): XML content that will be put between the <channel> element and the first <item> element for a RSS document, or between the xml tag and the first <entry> element for an Atom document. If it is specified, it will overload the following options.
- **TITLE=value:** (RSS or Atom document): value put inside the <title> element in the header. If not provided, a dummy value will be used as that element is compulsory.
- **DESCRIPTION=value:** (RSS document): value put inside the <description> element in the header. If not provided, a dummy value will be used as that element is compulsory.
- **LINK=value:** (RSS document): value put inside the <link> element in the header. If not provided, a dummy value will be used as that element is compulsory.
- **UPDATED=value:** (Atom document): value put inside the <updated> element in the header. Should be formatted as a XML datetime. If not provided, a dummy value will be used as that element is compulsory.
- **AUTHOR_NAME=value:** (Atom document): value put inside the <author><name> element in the header. If not provided, a dummy value will be used as that element is compulsory.
- **ID=value:** (Atom document): value put inside the <id> element in the header. If not provided, a dummy value will be used as that element is compulsory.

When translating from a source dataset, it may be necessary to rename the field names from the source dataset to the expected RSS or ATOM attribute names, such as <title>, <description>, etc... This can be done with a *OGR VRT* dataset, or by using the "-sql" option of the ogr2ogr utility (see rfc-21)

7.25.6 VSI Virtual File System API support

The driver supports reading and writing to files managed by VSI Virtual File System API, which include "regular" files, as well as files in the /vsizip/ (read-write) , /vsigzip/ (read-write) , /vsicurl/ (read-only) domains.

Writing to /dev/stdout or /vsistdout/ is also supported.

7.25.7 Examples

7.25.7.1 Example 1: Dump the contents of a GeoRSS datafile

```
ogrinfo -ro -al input.xml
```

7.25.7.2 Example 2: Translating an Atom document into an RSS document

Note: in this example we map equivalent fields, from the source name to the expected name of the destination format.

```
ogr2ogr -f GeoRSS output.xml input.xml "select link_href as link, title, content as␣
↪description, author_name as author, id as guid from georss"
```

7.25.7.3 Example 3: Reading the contents of a online GeoRSS feed using Python

```
#!/usr/bin/python
import gdal
import ogr
import urllib2

url = 'http://earthquake.usgs.gov/eqcenter/catalogs/eqs7day-M5.xml'
content = None
try:
    handle = urllib2.urlopen(url)
    content = handle.read()
except urllib2.HTTPError, e:
    print 'HTTP service for %s is down (HTTP Error: %d)' % (url, e.code)
except Exception:
    print 'HTTP service for %s is down.' %(url)

# Create in-memory file from the downloaded content
gdal.FileFromMemBuffer('/vsimem/temp', content)

ds = ogr.Open('/vsimem/temp')
lyr = ds.GetLayer(0)
feat = lyr.GetNextFeature()
while feat is not None:
    print feat.GetFieldAsString('title') + ' ' + feat.GetGeometryRef().ExportToWkt()
    feat.Destroy()
    feat = lyr.GetNextFeature()

ds.Destroy()

# Free memory associated with the in-memory file
gdal.Unlink('/vsimem/temp')
```

7.25.8 See Also

- Home page for GeoRSS format
- Wikipedia page for GeoRSS format
- Wikipedia page for RSS format
- RSS 2.0 specification

- Wikipedia page for Atom format
- Atom 1.0 specification

7.26 GML - Geography Markup Language

Driver short name

GML

Build dependencies

(read support needs Xerces or libexpat)

OGR has limited support for GML reading and writing. Update of existing files is not supported.

Supported GML flavors :

Read	Write
GML2 and GML3 that can be translated into simple feature model	GML 2.1.2 or GML 3 SF-0 (GML 3.1.1 Compliance level SF-0)

Another driver, *GMLAS*, for GML driven by application schemas, is also available. Both GML and GMLAS drivers have their use cases.

7.26.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.26.2 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). The following configuration options are available:

- **GML_PARSER=[EXPAT/XERCES]:** Select the XML parser. See *Parsers*.

- **GML_INVERT_AXIS_ORDER_IF_LAT_LONG=[YES/NO]:** See *CRS support*. Equivalent of *INVERT_AXIS_ORDER_IF_LAT_LONG* open option.
- **GML_CONSIDER_EPSG_AS_URN=[YES/NO]:** See *CRS support*. Equivalent of *CONSIDER_EPSG_AS_URN* open option.
- **GML_SWAP_COORDINATES=[AUTO/YES/NO]:** Defaults to AUTO. See *CRS support*. Equivalent of *SWAP_COORDINATES* open option.
- **GML_FIELDTYPES=ALWAYS_STRING:** If set to ALWAYS_STRING, treat all fields as strings instead of scanning values to detect field types. See *Schema*.
- **GML_ATTRIBUTES_TO_OGR_FIELDS=[YES/NO]:** If YES, create fields from attributes of GML elements. See *Schema*.
- **GML_GFS_TEMPLATE=<filename>:** Equivalent of *GFS_TEMPLATE*. See *Schema*.
- **GML_GET_SECONDARY_GEOM=[YES/NO]:** Retrieve node elements of TopoCurve geometries instead of edges. See *Geometry reading*.
- **GML_SKIP_RESOLVE_ELEMS=[NONE/ALL/HUGE/<list>]:** Defaults to ALL. Control the gml:xlink resolving. See *gml:xlink resolving*.
- **GML_SKIP_CORRUPTED_FEATURES=[YES/NO]:** Defaults to NO. If set to **YES**, skip features that cannot be parsed correctly. If set to **NO**, the driver will fail with an error when it encounters a feature that cannot be parsed correctly.
- **GML_SAVE_RESOLVED_TO=[SAME/<filename>]:** Control where file resolved by gml:xlink is saved. See *gml:xlink resolving*.
- **GML_FACE_HOLE_NEGATIVE=[YES/NO]:** Defaults to NO. Select interpretation rule for TopoSurfaces. See *TopoSurface interpretation rules [polygons and internal holes]*.
- **GML_EXPOSE_FID=[YES/NO]:** Equivalent of *EXPOSE_FID*.
- **GML_EXPOSE_GML_ID=[YES/NO]:** Equivalent of *EXPOSE_GML_ID*.
- **GML_READ_MODE=[AUTO/STANDARD/SEQUENTIAL_LAYERS/INTERLEAVED_LAYERS]:** Equivalent of *READ_MODE*. See *Performance issues with large multi-layer GML files.*.
- **GML_DOWNLOAD_SCHEMA=[YES/NO]:** (GDAL >= 3.10) Equivalent of *DOWNLOAD_SCHEMA*.
- **GML_USE_SCHEMA_IMPORT=[YES/NO]:** Equivalent of *USE_SCHEMA_IMPORT*.

7.26.3 Parsers

The reading part of the driver only works if OGR is built with Xerces linked in. When Xerces is unavailable, read support also works if OGR is built with Expat linked in. XML validation is disabled by default. GML writing is always supported, even without Xerces or Expat.

Note: if both Xerces and Expat are available at build time, the GML driver will preferentially select at runtime the Expat parser for cases where it is possible (GML file in a compatible encoding), and default back to Xerces parser in other cases. However, the choice of the parser can be overridden by specifying the *GML_PARSER* configuration option to **EXPAT** or **XERCES**.

7.26.4 CRS support

The GML driver has coordinate system support. This is only reported when all the geometries of a layer have a srsName attribute, whose value is the same for all geometries. For srsName such as "urn:ogc:def:crs:EPSG:" or "http://www.opengis.net/def/crs/EPSG/0/", for geographic coordinate systems (as returned by WFS 1.1.0 for example), the axis order should be (latitude, longitude) as required by the standards, but this is unusual and can cause issues with applications unaware of axis order. So by default, the driver will swap the coordinates so that they are in the (longitude,

latitude) order and report a SRS without axis order specified. It is possible to get the original (latitude, longitude) order and SRS with axis order by setting the configuration option *GML_INVERT_AXIS_ORDER_IF_LAT_LONG* to **NO**.

There also situations where the srsName is of the form "EPSG:XXXX" (whereas "urn:ogc:def:crs:EPSG::XXXX" would have been more explicit on the intent) and the coordinates in the file are in (latitude, longitude) order. By default, OGR will not consider the EPSG axis order and will report the coordinates in (latitude,longitude) order. However, if you set the configuration option *GML_CONSIDER_EPSG_AS_URN* to **YES**, the rules explained in the previous paragraph will be applied.

The above also applied for projected coordinate systems whose EPSG preferred axis order is (northing, easting).

The *SWAP_COORDINATES* open option (or *GML_SWAP_COORDINATES* configuration option) can be set to AUTO/YES/NO. It controls whether the order of the x/y or long/lat coordinates should be swapped. In AUTO mode, the driver will determine if swapping must be done from the srsName and value of other options like *CONSIDER_EPSG_AS_URN* and *INVERT_AXIS_ORDER_IF_LAT_LONG*. When *SWAP_COORDINATES* is set to YES, coordinates will be always swapped regarding the order they appear in the GML, and when it set to NO, they will be kept in the same order. The default is AUTO.

7.26.5 Schema

In contrast to most GML readers, the OGR GML reader does not require the presence of an XML Schema definition of the feature classes (file with .xsd extension) to be able to read the GML file. If the .xsd file is absent or OGR is not able to parse it, the driver attempts to automatically discover the feature classes and their associated properties by scanning the file and looking for "known" gml objects in the gml namespace to determine the organization. While this approach is error prone, it has the advantage of working for GML files even if the associated schema (.xsd) file has been lost.

It is possible to specify an explicit filename for the XSD schema to use, by using "a_filename.gml,xsd=another_filename.xsd" as a connection string. The XSD can also be specified as the value of the XSD open option.

The first time a GML file is opened, if the associated .xsd is absent or could not been parsed correctly, it is completely scanned in order to determine the set of featuretypes, the attributes associated with each and other dataset level information. This information is stored in a .gfs file with the same basename as the target gml file. Subsequent accesses to the same GML file will use the .gfs file to predefine dataset level information accelerating access. To a limited extent the .gfs file can be manually edited to alter how the GML file will be parsed. Be warned that the .gfs file will be ignored if the associated .gml file has a newer timestamp.

When prescanning the GML file to determine the list of feature types, and fields, the contents of fields are scanned to try and determine the type of the field. In some applications it is easier if all fields are just treated as string fields. This can be accomplished by setting the configuration option *GML_FIELDTYPES* to the value **ALWAYS_STRING**.

The *GML_ATTRIBUTES_TO_OGR_FIELDS* configuration option can be set to **YES** so that attributes of GML elements are also taken into account to create OGR fields.

You can use *GML_GFS_TEMPLATE* configuration option (or *GFS_TEMPLATE* open option) set to a **path_to_template.gfs** in order to unconditionally use a predefined GFS file. This option is really useful when you are planning to import many distinct GML files in subsequent steps [**-append**] and you absolutely want to preserve a fully consistent data layout for the whole GML set. Please, pay attention not to use the **-lco LAUNDER=yes** setting when using *GML_GFS_TEMPLATE*; this should break the correct recognition of attribute names between subsequent GML import runs.

7.26.6 Particular GML application schemas

Feature attributes in nested GML elements (non-flat attribute hierarchy) that can be found in some GML profiles, such as UK Ordnance Survey MasterMap, are detected. IntegerList, RealList and StringList field types when a GML element has several occurrences are also supported.

A specialized GML driver - the *NAS* driver - is available to read German AAA GML Exchange Format (NAS/ALKIS).

The GML driver has partial support for reading AIXM or CityGML files.

The GML driver supports reading :

- Finnish National Land Survey GML files (a.k.a MTK GML) for topographic data.
- Finnish National Land Survey GML files for cadastral data.
- Cadastral data in Inspire GML schemas.
- Czech RUIAN Exchange Format (VFR).

The GML driver supports reading responses to CSW GetRecords queries.

Since OGR 2.2, the GML driver supports reading Japanese FGD GML v4 files.

7.26.7 Geometry reading

When reading a feature, the driver will by default only take into account the last recognized GML geometry found (in case they are multiples) in the XML subtree describing the feature.

But, if the .xsd schema is understood by the XSD parser and declares several geometry fields, or the .gfs file declares several geometry fields, multiple geometry fields will be reported by the GML driver according to rfc-41.

In case of multiple geometry occurrences, if a geometry is in a <geometry> element, this will be the one selected. This will make default behavior consistent with Inspire objects.

The user can change the .gfs file to select the appropriate geometry by specifying its path with the <GeometryElementPath> element. See the description of the .gfs syntax below.

GML geometries including TopoCurve, TopoSurface, MultiCurve are also supported. The TopoCurve type GML geometry can be interpreted as either of two types of geometries. The Edge elements in it contain curves and their corresponding nodes. By default only the curves, the main geometries, are reported as OGRMultiLineString. To retrieve the nodes, as OGRMultiPoint, the configuration option *GML_GET_SECONDARY_GEOM* should be set to the value **YES**. When this is set only the secondary geometries are reported.

Arc, ArcString, ArcByBulge, ArcByCenterPoint, Circle and CircleByCenterPoints will be returned as circular string OGR geometries. If they are included in other GML elements such as CurveComposite, MultiCurve, Surface, corresponding non-linear OGR geometries will be returned as well. When reading GML3 application schemas, declarations of geometry fields such as CurvePropertyType, SurfacePropertyType, MultiCurvePropertyType or MultiSurfacePropertyType will be also interpreted as being potential non-linear geometries, and corresponding OGR geometry type will be used for the layer geometry type.

7.26.8 gml:xlink resolving

gml:xlink resolving is supported. When the resolver finds an element containing the tag xlink:href, it tries to find the corresponding element with the gml:id in the same gml file, other gml file in the file system or on the web using cURL. Set the configuration option *GML_SKIP_RESOLVE_ELEMS* to **NONE** to enable resolution.

By default the resolved file will be saved in the same directory as the original file with the extension ".resolved.gml", if it doesn't exist already. This behavior can be changed using the configuration option *GML_SAVE_RESOLVED_TO*. Set it to **SAME** to overwrite the original file. Set it to a **filename ending with .gml** to save it to that location. Any other values are ignored. If the resolver cannot write to the file for any reason, it will try to save it to a temporary file generated using CPLGenerateTempFilename("ResolvedGML"); if it cannot, resolution fails.

Note that the resolution algorithm is not optimized for large files. For files with more than a couple of thousand xlink:href tags, the process can go beyond a few minutes. A rough progress is displayed through CPLDebug() for every 256 links. It can be seen by setting the environment variable *CPL_DEBUG*. The resolution time can be reduced if you know any elements that will not be needed. Mention a comma separated list of names of such elements with the configuration option *GML_SKIP_RESOLVE_ELEMS*. Set it to **ALL** to skip resolving altogether (default action). Set it to **NONE** to resolve all the xlinks.

An alternative resolution method is available. This alternative method will be activated using the configuration option *GML_SKIP_RESOLVE_ELEMS=HUGE*. In this case any gml:xlink will be resolved using a temporary SQLite DB so to identify any corresponding gml:id relation. At the end of this SQL-based process, a resolved file will be generated exactly as in the **NONE** case but without their limits. The main advantages in using an external (temporary) DBMS so to resolve gml:xlink and gml:id relations are the following:

- no memory size constraints. The **NONE** method stores the whole GML node-tree in-memory; and this practically means that no GML file bigger than 1 GB can be processed at all using a 32-bit platform, due to memory allocation limits. Using a file-system based DBMS avoids at all this issue.
- by far better efficiency, most notably when huge GML files containing many thousands (or even millions) of xlink:href / gml:id relational pairs.
- using the *GML_SKIP_RESOLVE_ELEMS=HUGE* method realistically allows to successfully resolve some really huge GML file (3GB+) containing many millions xlink:href / gml:id in a reasonable time (about an hour or so on).
- The *GML_SKIP_RESOLVE_ELEMS=HUGE* method supports the following further configuration option:

7.26.9 TopoSurface interpretation rules [polygons and internal holes]

The GML driver is able to recognize two different interpretation rules for TopoSurface when a polygon contains any internal hole:

- the previously supported interpretation rule assumed that:
  - each TopoSurface may be represented as a collection of many Faces
  - *positive* Faces [i.e. declaring **orientation="+"**] are assumed to represent the Exterior Ring of some Polygon.
  - *negative* Faces [i.e. declaring **orientation="-"**] are assumed to represent an Interior Ring (aka *hole*) belonging to the latest declared Exterior Ring.
  - ordering any Edge used to represent each Ring is important: each Edge is expected to be exactly adjacent to the next one.
- the new interpretation rule now assumes that:
  - each TopoSurface may be represented as a collection of many Faces
  - the declared **orientation** for any Face has nothing to deal with Exterior/Interior Rings
  - each Face is now intended to represent a complete Polygon, eventually including any possible Interior Ring (*holes*)
  - the relative ordering of any Edge composing the same Face is completely not relevant

The newest interpretation seems to fully match GML 3 standard recommendations; so this latest is now assumed to be the default interpretation supported by OGR.

NOTE : Using the newest interpretation requires GDAL/OGR to be built against the GEOS library.

Using the *GML_FACE_HOLE_NEGATIVE* configuration option you can select the actual interpretation to be applied when parsing GML 3 topologies:

- setting *GML_FACE_HOLE_NEGATIVE=NO* (*default* option) will activate the newest interpretation rule
- but explicitly setting *GML_FACE_HOLE_NEGATIVE=YES* still enables to activate the old interpretation rule

7.26.10 Encoding issues

Expat library supports reading the following built-in encodings :

- US-ASCII
- UTF-8
- UTF-16
- ISO-8859-1
- Windows-1252

The content returned by OGR will be encoded in UTF-8, after the conversion from the encoding mentioned in the file header is.

If the GML file is not encoded in one of the previous encodings and the only parser available is Expat, it will not be parsed by the GML driver. You may convert it into one of the supported encodings with the *iconv* utility for example and change accordingly the *encoding* parameter value in the XML header.

When writing a GML file, the driver expects UTF-8 content to be passed in.

Note: The .xsd schema files are parsed with an integrated XML parser which does not currently understand XML encodings specified in the XML header. It expects encoding to be always UTF-8. If attribute names in the schema file contains non-ascii characters, it is better to use *iconv* utility and convert the .xsd file into UTF-8 encoding first.

7.26.11 Feature id (fid / gml:id)

The driver exposes the content of the gml:id attribute as a string field called *gml_id*, when reading GML WFS documents. When creating a GML3 document, if a field is called *gml_id*, its content will also be used to write the content of the gml:id attribute of the created feature.

The driver autodetects the presence of a fid (GML2) (resp. gml:id (GML3)) attribute at the beginning of the file, and, if found, exposes it by default as a *fid* (resp. *gml_id*) field. The autodetection can be overridden by specifying the `GML_EXPOSE_FID` or `GML_EXPOSE_GML_ID` configuration option to **YES** or **NO**.

When creating a GML2 document, if a field is called *fid*, its content will also be used to write the content of the fid attribute of the created feature.

7.26.12 Performance issues with large multi-layer GML files.

There is only one GML parser per GML datasource shared among the various layers. By default, the GML driver will restart reading from the beginning of the file, each time a layer is accessed for the first time, which can lead to poor performance with large GML files.

The `GML_READ_MODE` configuration option can be set to **SEQUENTIAL_LAYERS** if all features belonging to the same layer are written sequentially in the file. The reader will then avoid unnecessary resets when layers are read completely one after the other. To get the best performance, the layers must be read in the order they appear in the file.

If no .xsd and .gfs files are found, the parser will detect the layout of layers when building the .gfs file. If the layers are found to be sequential, a *<SequentialLayers>true</SequentialLayers>* element will be written in the .gfs file, so that the `GML_READ_MODE` will be automatically initialized to SEQUENTIAL_LAYERS if not explicitly set by the user.

The `GML_READ_MODE` configuration option can be set to INTERLEAVED_LAYERS to be able to read a GML file whose features from different layers are interleaved. In the case, the semantics of the GetNextFeature() will be slightly altered, in a way where a NULL return does not necessarily mean that all features from the current layer have been read, but it could also mean that there is still a feature to read, but that belongs to another layer. In that case, the file should be read with code similar to the following one :

```
int nLayerCount = poDS->GetLayerCount();
int bFoundFeature;
do
{
    bFoundFeature = FALSE;
    for( int iLayer = 0; iLayer < nLayerCount; iLayer++ )
    {
        OGRLayer   *poLayer = poDS->GetLayer(iLayer);
        OGRFeature *poFeature;
        while((poFeature = poLayer->GetNextFeature()) != NULL)
        {
            bFoundFeature = TRUE;
            poFeature->DumpReadable(stdout, NULL);
            OGRFeature::DestroyFeature(poFeature);
        }
    }
} while (bInterleaved && bFoundFeature);
```

7.26.13 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are supported:

- **XSD=<filename>:** to specify an explicit filename for the XSD application schema to use.
- **WRITE_GFS=[AUTO/YES/NO]:** (GDAL >= 3.1) whether to write a .gfs file. In AUTO mode, the .gfs file is only written if there is no recognized .xsd file, no existing .gfs file and for non-network file systems. This option can be set to YES for force .gfs file writing in situations where AUTO would not attempt to do it. Or it can be set to NO to disable .gfs file writing.
- **GFS_TEMPLATE=<filename>:** to unconditionally use a predefined GFS file. This option is really useful when you are planning to import many distinct GML files in subsequent steps [**-append**] and you absolutely want to preserve a fully consistent data layout for the whole GML set. Please, pay attention not to use the **-lco LAUNDER=yes** setting when this option; this should break the correct recognition of attribute names between subsequent GML import runs.
- **FORCE_SRS_DETECTION=[YES/NO]:** Defaults to `NO`. Force a full scan to detect the SRS of layers. This option may be needed in the case where the .gml file is accompanied with a .xsd. Normally in that situation, OGR would not detect the SRS, because this requires to do a full scan of the file.
- **EMPTY_AS_NULL=[YES/NO]:** Defaults to `YES`. If YES, fields with empty content will be reported as being NULL, instead of being an empty string. This is the historic behavior. However this will prevent such fields to be declared as not-nullable if the application schema declared them as mandatory. So this option can be set to NO to have both empty strings being report as such, and mandatory fields being reported as not nullable.
- **GML_ATTRIBUTES_TO_OGR_FIELDS=[YES/NO]:** Defaults to `NO`. Whether GML attributes should be reported as OGR fields. Note that this option has only an effect the first time a GML file is opened (before the .gfs file is created), and if it has no valid associated .xsd.
- **INVERT_AXIS_ORDER_IF_LAT_LONG=[YES/NO]:** Defaults to `YES`. Whether to present SRS and coordinate ordering in traditional GIS order.
- **CONSIDER_EPSG_AS_URN=[YES/NO/AUTO]:** Defaults to `AUTO`. Whether to consider srsName like EPSG:XXXX as respecting EPSG axis order.
- **SKIP_CORRUPTED_FEATURES=[YES/NO]:** Defaults to `NO`. If set to **YES**, skip features that cannot be parsed correctly. If set to **NO**, the driver will fail with an error when it encounters a feature that cannot be parsed correctly.

- **SWAP_COORDINATES=[AUTO/YES/NO]:** Defaults to `AUTO`. Whether the order of the x/y or long/lat coordinates should be swapped. In AUTO mode, the driver will determine if swapping must be done from the srsName and value of other options like *`CONSIDER_EPSG_AS_URN`* and *`INVERT_AXIS_ORDER_IF_LAT_LONG`*. When *`SWAP_COORDINATES`* is set to YES, coordinates will be always swapped regarding the order they appear in the GML, and when it set to NO, they will be kept in the same order.
- **SKIP_RESOLVE_ELEMS=[NONE/ALL/HUGE/<list>]:** Defaults to `ALL`. Control the gml:xlink resolving. See *gml:xlink resolving*.
- **READ_MODE=[AUTO/STANDARD/SEQUENTIAL_LAYERS/INTERLEAVED_LAYERS]:** Defaults to `AUTO`. Read mode.
- **EXPOSE_GML_ID=[YES/NO/AUTO]:** Defaults to `AUTO`. Whether to make feature gml:id as a gml_id attribute.
- **EXPOSE_FID=[YES/NO/AUTO]:** Defaults to `AUTO`. Whether to make feature fid as a fid attribute.
- **DOWNLOAD_SCHEMA=[YES/NO]:** Defaults to `YES`. Whether to download the remote application schema if needed (only if the document looks like a WFS response currently).
- **REGISTRY=<filename>:** Defaults to `{GDAL_DATA}/gml_registry.xml.`. Filename of the registry with application schemas.
- **USE_BBOX=[YES/NO]:** (GDAL >= 3.7.1) Defaults to `NO`. Whether to use gml:boundedBy at feature level as feature geometry, if there are no other geometry.
- **USE_SCHEMA_IMPORT=[YES/NO]:** Defaults to `NO`. Whether to use schema imports in XSD files so that the feature types corresponding to imported schema can be detected.
- **OGR_SCHEMA=<filename>|<json string>:** (GDAL >= 3.11.0) Partially or totally overrides the auto-detected schema to use for creating the layer. The overrides are applied to the schema detected from the GML file or the *.xsd* or the *.gfs* ` file if present. The overrides are defined as a JSON list of field definitions. This can be a filename, a URL or JSON string conformant with the ogr_fields_override.schema.json schema

> **Note**
>
> When changing the value of most of the above options, it is recommended to delete the `.gfs` file if it pre-exists, otherwise mis-behavior might be observed.

7.26.14 Creation Issues

On export all layers are written to a single GML file all in a single feature collection. Each layer's name is used as the element name for objects from that layer. Geometries are always written as the ogr:geometryProperty element on the feature.

7.26.15 Dataset creation options

Dataset creation options can be specified in command-line tools using the syntax `-dsco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following dataset creation options are supported:

- **XSISCHEMAURI=value:** If provided, this URI will be inserted as the schema location. Note that the schema file isn't actually accessed by OGR, so it is up to the user to ensure it will match the schema of the OGR produced GML data file.
- **XSISCHEMA=[EXTERNAL/INTERNAL/OFF]:** Defaults to `EXTERNAL`. If EXTERNAL. This writes a GML application schema file to a corresponding .xsd file (with the same basename). If INTERNAL is used the schema is written within the GML file, but this is experimental and almost certainly not valid XML. OFF disables schema generation (and is implicit if *`XSISCHEMAURI`* is used).

- **PREFIX=value:** Defaults to `ogr`. This is the prefix for the application target namespace.
- **STRIP_PREFIX=[TRUE/FALSE]:** Defaults to `FALSE`. Can be set to TRUE to avoid writing the prefix of the application target namespace in the GML file.
- **TARGET_NAMESPACE=value:** Defaults to `http://ogr.maptools.org/`. This is the application target namespace.
- **FORMAT=[GML2/GML3/GML3Deegree/GMl3.2]:** Select from the following formats:
  - *GML2* in order to write GML files that follow GML 2.1.2 (Default before GDAL 3.4)
  - *GML3* in order to write GML files that follow GML 3.1.1 SF-0 profile.
  - *GML3Deegree* in order to produce a GML 3.1.1 .XSD schema, with a few variations with respect to what is recommended by GML3 SF-0 profile, but that will be better accepted by some software (such as Deegree 3).
  - *GML3.2*in order to write GML files that follow GML 3.2.1 SF-0 profile. (Default since GDAL 3.4)

  Non-linear geometries can be written. This is only compatible with selecting on of that above GML3 format variant. Otherwise, such geometries will be approximating into their closest matching linear geometry. Note: fields of type StringList, RealList or IntegerList can be written. This will cause to advertise the SF-1 profile in the .XSD schema (such types are not supported by SF-0).
- **GML_FEATURE_COLLECTION=[YES/NO]:** Whether to use the gml:FeatureCollection, instead of creating a dedicated container element in the target namespace. Only valid for FORMAT=GML3/GML3.2. Note that gml:FeatureCollection has been deprecated in GML 3.2, and is not allowed by the OGC 06-049r1 "Geography Markup Language (GML) simple features profile" (for GML 3.1.1) and OGC 10-100r3 "Geography Markup Language (GML) simple features profile (with Corrigendum)" (for GML 3.2) specifications.
- **GML3_LONGSRS=[YES/NO]:** (only valid when *`FORMAT=GML3/GML3Degree/GML3.2`*) Deprecated by *`SRSNAME_FORMAT`* in GDAL 2.2. Default to YES. If YES, SRS with EPSG authority will be written with the "urn:ogc:def:crs:EPSG::" prefix. In the case the SRS is a SRS without explicit AXIS order, but that the same SRS authority code imported with ImportFromEPSGA() should be treated as lat/long or northing/easting, then the function will take care of coordinate order swapping. If set to NO, SRS with EPSG authority will be written with the "EPSG:" prefix, even if they are in lat/long order.
- **SRSNAME_FORMAT=[SHORT/OGC_URN/OGC_URL]:** Defaults to `OGC_URN`. (Only valid for *`FORMAT=GML3/GML3Degree/GML3.2`*). If SHORT, then srsName will be in the form AUTHORITY_NAME:AUTHORITY_CODE If OGC_URN, then srsName will be in the form urn:ogc:def:crs:AUTHORITY_NAME::AUTHORITY_CODE If OGC_URL, then srsName will be in the form http://www.opengis.net/def/crs/AUTHORITY_NAME/0/AUTHORITY_CODE For OGC_URN and OGC_URL, in the case the SRS is a SRS without explicit AXIS order, but that the same SRS authority code imported with ImportFromEPSGA() should be treated as lat/long or northing/easting, then the function will take care of coordinate order swapping.
- **SRSDIMENSION_LOC=[POSLIST/GEOMETRY/GEOMETRY,POSLIST]:** (Only valid for *`FORMAT=GML3/GML3Degree/GML3.2`*) Default to POSLIST. For 2.5D geometries, define the location where to attach the srsDimension attribute. There are diverging implementations. Some put in on the <gml:posList> element, other on the top geometry element.
- **WRITE_FEATURE_BOUNDED_BY=[YES/NO]:** Defaults to `YES`. (only valid when *`FORMAT=GML3/GML3Degree/GML3.2`*) If set to NO, the <gml:boundedBy> element will not be written for each feature.
- **SPACE_INDENTATION=[YES/NO]:** Defaults to `YES`. If YES, the output will be indented with spaces for more readability, but at the expense of file size.
- **GML_ID=value:** Defaults to `aFeatureCollection`. (Only valid for GML 3.2) Value of feature collection gml:id.
- **NAME=value:** Content of GML name element. Can also be set as the NAME metadata item on the dataset.

- **DESCRIPTION=value:** Content of GML description element. Can also be set as the DESCRIPTION metadata item on the dataset.

7.26.16 VSI Virtual File System API support

The driver supports reading and writing to files managed by VSI Virtual File System API, which include "regular" files, as well as files in the /vsizip/ (read-write) , /vsigzip/ (read-write) , /vsicurl/ (read-only) domains.

Writing to /dev/stdout or /vsistdout/ is also supported. Note that in that case, only the content of the GML file will be written to the standard output (and not the .xsd). The <boundedBy> element will not be written. This is also the case if writing in /vsigzip/

7.26.17 Syntax of .gfs files

A XML Schema for .gfs files can be found at https://raw.githubusercontent.com/OSGeo/gdal/master/ogr/ogrsf_frmts/gml/data/gfs.xsd .

Let's consider the following test.gml file :

```
<?xml version="1.0" encoding="UTF-8"?>
<gml:FeatureCollection xmlns:gml="http://www.opengis.net/gml">
  <gml:featureMember>
    <LAYER>
      <attrib1>attrib1_value</attrib1>
      <attrib2container>
        <attrib2>attrib2_value</attrib2>
      </attrib2container>
      <location1container>
        <location1>
            <gml:Point><gml:coordinates>3,50</gml:coordinates></gml:Point>
        </location1>
      </location1container>
      <location2>
        <gml:Point><gml:coordinates>2,49</gml:coordinates></gml:Point>
      </location2>
    </LAYER>
  </gml:featureMember>
</gml:FeatureCollection>
```

and the following associated .gfs file.

```
<GMLFeatureClassList>
  <GMLFeatureClass>
    <Name>LAYER</Name>
    <ElementPath>LAYER</ElementPath>
    <GeometryElementPath>location1container|location1</GeometryElementPath>
    <PropertyDefn>
      <Name>attrib1</Name>
      <ElementPath>attrib1</ElementPath>
      <Type>String</Type>
      <Width>13</Width>
    </PropertyDefn>
    <PropertyDefn>
      <Name>attrib2</Name>
      <ElementPath>attrib2container|attrib2</ElementPath>
```

(continues on next page)

(continued from previous page)

```
      <Type>String</Type>
      <Width>13</Width>
    </PropertyDefn>
  </GMLFeatureClass>
</GMLFeatureClassList>
```

Note the presence of the '|' character in the <ElementPath> and <GeometryElementPath> elements to specify the wished field/geometry element that is a nested XML element. Nested field elements are supported, as well as specifying <GeometryElementPath> If GeometryElementPath is not specified, the GML driver will use the last recognized geometry element.

The <GeometryType> element can be specified to force the geometry type. Accepted values are : 0 (any geometry type), 1 (point), 2 (linestring), 3 (polygon), 4 (multipoint), 5 (multilinestring), 6 (multipolygon), 7 (geometrycollection).

The <GeometryElementPath> and <GeometryType> can be specified as many times as there are geometry fields in the GML file. Another possibility is to define a <GeomPropertyDefn>element as many times as necessary:

```
<GMLFeatureClassList>
  <GMLFeatureClass>
    <Name>LAYER</Name>
    <ElementPath>LAYER</ElementPath>
    <GeomPropertyDefn>
        <Name>geometry</Name> <!-- OGR geometry name -->
        <ElementPath>geometry</ElementPath> <!-- XML element name possibly with '|' to␣
↪specify the path -->
        <Type>MultiPolygon</Type>
    </GeomPropertyDefn>
    <GeomPropertyDefn>
        <Name>referencePoint</Name>
        <ElementPath>referencePoint</ElementPath>
        <Type>Point</Type>
    </GeomPropertyDefn>
  </GMLFeatureClass>
</GMLFeatureClassList>
```

The output of *ogrinfo test.gml -ro -al* is:

```
Layer name: LAYER
Geometry: Unknown (any)
Feature Count: 1
Extent: (3.000000, 50.000000) - (3.000000, 50.000000)
Layer SRS WKT:
(unknown)
Geometry Column = location1container|location1
attrib1: String (13.0)
attrib2: String (13.0)
OGRFeature(LAYER):0
  attrib1 (String) = attrib1_value
  attrib2 (String) = attrib2_value
  POINT (3 50)
```

7.26.18 Advanced .gfs syntax

7.26.18.1 Specifying ElementPath to find objects embedded into top level objects

Let's consider the following test.gml file :

```
<?xml version="1.0" encoding="utf-8"?>
<gml:FeatureCollection xmlns:xlink="http://www.w3.org/1999/xlink"
                       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                       gml:id="foo" xmlns:gml="http://www.opengis.net/gml/3.2">
  <gml:featureMember>
    <TopLevelObject gml:id="TopLevelObject.1">
      <content>
        <Object gml:id="Object.1">
          <geometry>
            <gml:Polygon gml:id="Object.1.Geometry" srsName="urn:ogc:def:crs:EPSG::4326">
              <gml:exterior>
                <gml:LinearRing>
                  <gml:posList srsDimension="2">48 2 49 2 49 3 48 3 48 2</gml:posList>
                </gml:LinearRing>
              </gml:exterior>
            </gml:Polygon>
          </geometry>
          <foo>bar</foo>
        </Object>
      </content>
      <content>
        <Object gml:id="Object.2">
          <geometry>
            <gml:Polygon gml:id="Object.2.Geometry" srsName="urn:ogc:def:crs:EPSG::4326">
              <gml:exterior>
                <gml:LinearRing>
                  <gml:posList srsDimension="2">-48 2 -49 2 -49 3 -48 3 -48 2</
↪gml:posList>
                </gml:LinearRing>
              </gml:exterior>
            </gml:Polygon>
          </geometry>
          <foo>baz</foo>
        </Object>
      </content>
    </TopLevelObject>
  </gml:featureMember>
</gml:FeatureCollection>
```

By default, only the TopLevelObject object would be reported and it would only use the second geometry. This is not the desired behavior in that instance. You can edit the generated .gfs and modify it like the following in order to specify a full path to the element (top level XML element being omitted) :

```
<GMLFeatureClassList>
  <GMLFeatureClass>
    <Name>Object</Name>
    <ElementPath>featureMember|TopLevelObject|content|Object</ElementPath>
    <GeometryType>3</GeometryType>
```

(continues on next page)

(continued from previous page)

```
    <PropertyDefn>
      <Name>foo</Name>
      <ElementPath>foo</ElementPath>
      <Type>String</Type>
    </PropertyDefn>
  </GMLFeatureClass>
</GMLFeatureClassList>
```

7.26.18.2 Getting XML attributes as OGR fields

The element@attribute syntax can be used in the <ElementPath> to specify that the value of attribute 'attribute' of element 'element' must be fetched.

Let's consider the following test.gml file :

```
<?xml version="1.0" encoding="UTF-8"?>
<gml:FeatureCollection xmlns:gml="http://www.opengis.net/gml">
  <gml:featureMember>
    <LAYER>
      <length unit="m">5</length>
    </LAYER>
  </gml:featureMember>
</gml:FeatureCollection>
```

and the following associated .gfs file.

```
<GMLFeatureClassList>
  <GMLFeatureClass>
    <Name>LAYER</Name>
    <ElementPath>LAYER</ElementPath>
    <GeometryType>100</GeometryType> <!-- no geometry -->
    <PropertyDefn>
      <Name>length</Name>
      <ElementPath>length</ElementPath>
      <Type>Real</Type>
    </PropertyDefn>
    <PropertyDefn>
      <Name>length_unit</Name>
      <ElementPath>length@unit</ElementPath>
      <Type>String</Type>
    </PropertyDefn>
  </GMLFeatureClass>
</GMLFeatureClassList>
```

The output of *ogrinfo test.gml -ro -al* is:

```
Layer name: LAYER
Geometry: None
Feature Count: 1
Layer SRS WKT:
(unknown)
gml_id: String (0.0)
length: Real (0.0)
```

(continues on next page)

(continued from previous page)

```
length_unit: String (0.0)
OGRFeature(LAYER):0
  gml_id (String) = (null)
  length (Real) = 5
  length_unit (String) = m
```

7.26.18.3 Using conditions on XML attributes

A <Condition> element can be specified as a child element of a <PropertyDefn>. The content of the Condition follows a minimalistic XPath syntax. It must be of the form @attrname[=|!=]'attrvalue' [and|or other_cond]*. Note that 'and' and 'or' operators cannot be mixed (their precedence is not taken into account).

Several <PropertyDefn> can be defined with the same <ElementPath>, but with <Condition> that must be mutually exclusive.

Let's consider the following testcondition.gml file :

```
<?xml version="1.0" encoding="utf-8" ?>
<ogr:FeatureCollection
     xmlns:ogr="http://ogr.maptools.org/"
     xmlns:gml="http://www.opengis.net/gml">
  <gml:featureMember>
    <ogr:testcondition fid="testcondition.0">
      <ogr:name lang="en">English name</ogr:name>
      <ogr:name lang="fr">Nom francais</ogr:name>
      <ogr:name lang="de">Deutsche name</ogr:name>
    </ogr:testcondition>
  </gml:featureMember>
</ogr:FeatureCollection>
```

and the following associated .gfs file.

```
<GMLFeatureClassList>
  <GMLFeatureClass>
    <Name>testcondition</Name>
    <ElementPath>testcondition</ElementPath>
    <GeometryType>100</GeometryType>
    <PropertyDefn>
      <Name>name_en</Name>
      <ElementPath>name</ElementPath>
      <Condition>@lang='en'</Condition>
      <Type>String</Type>
    </PropertyDefn>
    <PropertyDefn>
      <Name>name_fr</Name>
      <ElementPath>name</ElementPath>
      <Condition>@lang='fr'</Condition>
      <Type>String</Type>
    </PropertyDefn>
    <PropertyDefn>
      <Name>name_others_lang</Name>
      <ElementPath>name@lang</ElementPath>
      <Condition>@lang!='en' and @lang!='fr'</Condition>
```

(continues on next page)

(continued from previous page)

```
      <Type>StringList</Type>
    </PropertyDefn>
    <PropertyDefn>
      <Name>name_others</Name>
      <ElementPath>name</ElementPath>
      <Condition>@lang!='en' and @lang!='fr'</Condition>
      <Type>StringList</Type>
    </PropertyDefn>
  </GMLFeatureClass>
</GMLFeatureClassList>
```

The output of *ogrinfo testcondition.gml -ro -al* is:

```
Layer name: testcondition
Geometry: None
Feature Count: 1
Layer SRS WKT:
(unknown)
fid: String (0.0)
name_en: String (0.0)
name_fr: String (0.0)
name_others_lang: StringList (0.0)
name_others: StringList (0.0)
OGRFeature(testcondition):0
  fid (String) = testcondition.0
  name_en (String) = English name
  name_fr (String) = Nom francais
  name_others_lang (StringList) = (1:de)
  name_others (StringList) = (1:Deutsche name)
```

7.26.19 Registry for GML application schemas

The "data" directory of the GDAL installation contains a "gml_registry.xml" file that links feature types of GML application schemas to .xsd or .gfs files that contain their definition. This is used in case no valid .gfs or .xsd file is found next to the GML file.

An alternate location for the registry file can be defined by setting its full pathname to the GML_REGISTRY configuration option.

An example of such a file is :

```
<gml_registry>
    <!-- Finnish National Land Survey cadastral data -->
    <namespace prefix="ktjkiiwfs" uri="http://xml.nls.fi/ktjkiiwfs/2010/02"␣
↪useGlobalSRSName="true">
        <featureType elementName="KiinteistorajanSijaintitiedot"
                 schemaLocation="http://xml.nls.fi/XML/Schema/sovellus/ktjkii/modules/
↪kiinteistotietojen_kyselypalvelu_WFS/Asiakasdokumentaatio/ktjkiiwfs/2010/02/
↪KiinteistorajanSijaintitiedot.xsd"/>
        <featureType elementName="PalstanTunnuspisteenSijaintitiedot"
                 schemaLocation="http://xml.nls.fi/XML/Schema/sovellus/ktjkii/modules/
↪kiinteistotietojen_kyselypalvelu_WFS/Asiakasdokumentaatio/ktjkiiwfs/2010/02/
↪palstanTunnuspisteenSijaintitiedot.xsd"/>
```

(continues on next page)

(continued from previous page)

```
        <featureType elementName="RekisteriyksikonTietoja"
                 schemaLocation="http://xml.nls.fi/XML/Schema/sovellus/ktjkii/modules/
↪kiinteistotietojen_kyselypalvelu_WFS/Asiakasdokumentaatio/ktjkiiwfs/2010/02/
↪RekisteriyksikonTietoja.xsd"/>
        <featureType elementName="PalstanTietoja"
                 schemaLocation="http://xml.nls.fi/XML/Schema/sovellus/ktjkii/modules/
↪kiinteistotietojen_kyselypalvelu_WFS/Asiakasdokumentaatio/ktjkiiwfs/2010/02/
↪PalstanTietoja.xsd"/>
    </namespace>

    <!-- Inspire CadastralParcels schema -->
    <namespace prefix="cp" uri="urn:x-inspire:specification:gmlas:CadastralParcels:3.0"␣
↪useGlobalSRSName="true">
        <featureType elementName="BasicPropertyUnit"
                     gfsSchemaLocation="inspire_cp_BasicPropertyUnit.gfs"/>
        <featureType elementName="CadastralBoundary"
                     gfsSchemaLocation="inspire_cp_CadastralBoundary.gfs"/>
        <featureType elementName="CadastralParcel"
                     gfsSchemaLocation="inspire_cp_CadastralParcel.gfs"/>
        <featureType elementName="CadastralZoning"
                     gfsSchemaLocation="inspire_cp_CadastralZoning.gfs"/>
    </namespace>

    <!-- Czech RUIAN (VFR) schema (v1) -->
    <namespace prefix="vf"
               uri="urn:cz:isvs:ruian:schemas:VymennyFormatTypy:v1 ../ruian/xsd/vymenny_
↪format/VymennyFormatTypy.xsd"
               useGlobalSRSName="true">
        <featureType elementName="TypSouboru"
                     elementValue="OB"
                     gfsSchemaLocation="ruian_vf_ob_v1.gfs"/>
        <featureType elementName="TypSouboru"
                     elementValue="ST"
                     gfsSchemaLocation="ruian_vf_st_v1.gfs"/>
    </namespace>
</gml_registry>
```

XML schema definition (.xsd) files are pointed by the schemaLocation attribute, whereas OGR .gfs files are pointed by the gfsSchemaLocation attribute. In both cases, the filename can be a URL (http://, https://), an absolute filename, or a relative filename (relative to the location of gml_registry.xml).

The schema is used if and only if the namespace prefix and URI are found in the first bytes of the GML file (e.g. *xmlns:ktjkiiwfs="http://xml.nls.fi/ktjkiiwfs/2010/02"*), and that the feature type is also detected in the first bytes of the GML file (e.g. *ktjkiiwfs:KiinteistorajanSijaintitiedot*). If the element value is defined than the schema is used only if the feature type together with the value is found in the first bytes of the GML file (e.g. *vf:TypSouboru>OB_UKSH*).

7.26.20 Building junction tables

The ogr_build_junction_table.py script can be used to build a junction table from OGR layers that contain "XXXX_href" fields. Let's considering the following output of a GML file with links to other features :

```
OGRFeature(myFeature):1
  gml_id (String) = myFeature.1
```

(continues on next page)

(continued from previous page)

```
  [...]
  otherFeature_href (StringList) = (2:#otherFeature.10,#otherFeature.20)

OGRFeature(myFeature):2
  gml_id (String) = myFeature.2
  [...]
  otherFeature_href (StringList) = (2:#otherFeature.30,#otherFeature.10)
```

After running

```
ogr2ogr -f PG PG:dbname=mydb my.gml
```

to import it into PostGIS and

```
python3 ogr_build_junction_table.py PG:dbname=mydb
```

, a *myfeature_otherfeature* table will be created and will contain the following content :

myfeature_gml_id	otherfeature_gml_id
myFeature.1	otherFeature.10
myFeature.1	otherFeature.20
myFeature.2	otherFeature.30
myFeature.2	otherFeature.10

7.26.21 Reading datasets resulting from a WFS 2.0 join queries

The GML driver can read datasets resulting from a WFS 2.0 join queries.

Such datasets typically look like:

```
<wfs:FeatureCollection xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:app="http://app.com"
    xmlns:wfs="http://www.opengis.net/wfs/2.0"
    xmlns:gml="http://www.opengis.net/gml/3.2"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    numberMatched="unknown" numberReturned="2" timeStamp="2015-01-01T00:00:00.000Z"
    xsi:schemaLocation="http://www.opengis.net/gml/3.2 http://schemas.opengis.net/gml/3.
↪2.1/gml.xsd
                        http://www.opengis.net/wfs/2.0 http://schemas.opengis.net/wfs/2.
↪0/wfs.xsd">
  <wfs:member>
    <wfs:Tuple>
      <wfs:member>
        <app:table1 gml:id="table1-1">
          <app:foo>1</app:foo>
        </app:table1>
      </wfs:member>
      <wfs:member>
        <app:table2 gml:id="table2-1">
          <app:bar>2</app:bar>
          <app:baz>foo</app:baz>
          <app:geometry><gml:Point gml:id="table2-2.geom.0"><gml:pos>2 49</gml:pos></
```

(continues on next page)

(continued from previous page)

```
↪gml:Point></app:geometry>
        </app:table2>
      </wfs:member>
    </wfs:Tuple>
  </wfs:member>
  <wfs:member>
    <wfs:Tuple>
      <wfs:member>
        <app:table1 gml:id="table1-2">
          <app:bar>2</app:bar>
          <app:geometry><gml:Point gml:id="table1-1.geom.0"><gml:pos>3 50</gml:pos></
↪gml:Point></app:geometry>
        </app:table1>
      </wfs:member>
      <wfs:member>
        <app:table2 gml:id="table2-2">
          <app:bar>2</app:bar>
          <app:baz>bar</app:baz>
          <app:geometry><gml:Point gml:id="table2-2.geom.0"><gml:pos>2 50</gml:pos></
↪gml:Point></app:geometry>
        </app:table2>
      </wfs:member>
    </wfs:Tuple>
  </wfs:member>
</wfs:FeatureCollection>
```

OGR will group together the attributes from the layers participating to the join and will prefix them with the layer name. So the above example will be read as the following:

```
OGRFeature(join_table1_table2):0
  table1.gml_id (String) = table1-1
  table1.foo (Integer) = 1
  table1.bar (Integer) = (null)
  table2.gml_id (String) = table2-1
  table2.bar (Integer) = 2
  table2.baz (String) = foo
  table2.geometry = POINT (2 49)

OGRFeature(join_table1_table2):1
  table1.gml_id (String) = table1-2
  table1.foo (Integer) = (null)
  table1.bar (Integer) = 2
  table2.gml_id (String) = table2-2
  table2.bar (Integer) = 2
  table2.baz (String) = bar
  table1.geometry = POINT (3 50)
  table2.geometry = POINT (2 50)
```

7.26.22 Geometry coordinate precision

Added in version GDAL: 3.9

The GML driver supports reading and writing the geometry coordinate precision, using the *OGRGeomCoordinatePrecision* settings of the *OGRGeomFieldDefn*. Those settings are used to round the coordinates of the geometry of the features to an appropriate decimal precision.

Implementation details: the coordinate precision is stored in the XML schema as `xs:annotation/xs:appinfo[source="http://ogr.maptools.org/"]/ogr:xy_coordinate_resolution` and `xs:annotation/xs:appinfo[source="http://ogr.maptools.org/"]/ogr:z_coordinate_resolution` optional elements in the declaration of the geometry column. Their numeric value is expressed in the units of the SRS.

Example:

```
<xs:element name="my_geom" type="gml:SurfacePropertyType" nillable="true" minOccurs="0"␣
↪maxOccurs="1">
  <xs:annotation>
    <xs:appinfo source="http://ogr.maptools.org/">
      <ogr:xy_coordinate_resolution>8.9e-8</ogr:xy_coordinate_resolution>
      <ogr:z_coordinate_resolution>1e-3</ogr:z_coordinate_resolution>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

7.26.23 Examples

7.26.23.1 Example 1: Writing the results of an Oracle query to GML

```
ogr2ogr -f GML output.gml OCI:usr/pwd@db my_feature -where "id = 0"
```

7.26.23.2 Example 2: Writing the results of a PostGIS query to GML

```
ogr2ogr -f GML output.gml PG:'host=myserver dbname=warmerda' -sql "SELECT pop_1994 from␣
↪canada where province_name = 'Alberta'"
```

7.26.24 See Also

- GML Specifications
- GML 3.1.1 simple features profile - OGC(R) 06-049r1
- Geography Markup Language (GML) simple features profile (with Corrigendum) (GML 3.2.1) - OGC(R) 10-100r3
- Xerces
- *GMLAS - Geography Markup Language (GML) driven by application schemas*
- *NAS/ALKIS : specialized GML driver for cadastral data in Germany*

7.26.25 Credits

- Implementation for *GML_SKIP_RESOLVE_ELEMS=HUGE* was contributed by A.Furieri, with funding from Regione Toscana
- Support for cadastral data in Finnish National Land Survey GML and Inspire GML was funded by The Information Centre of the Ministry of Agriculture and Forestry (Tike)

7.27 GMLAS - Geography Markup Language (GML) driven by application schemas

Driver short name

GMLAS

Build dependencies

Xerces

This driver can read and write XML files of arbitrary structure, included those containing so called Complex Features, provided that they are accompanied by one or several XML schemas that describe the structure of their content. While this driver is generic to any XML schema, the main target is to be able to read and write documents referencing directly or indirectly to the GML namespace.

The driver requires Xerces-C >= 3.1.

The driver can deal with files of arbitrary size with a very modest RAM usage, due to its working in streaming mode.

7.27.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports CreateCopy()

This driver supports the *`GDALDriver::CreateCopy()`* operation

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.27.2 Opening syntax

The connection string is GMLAS:/path/to/the.gml. Note the `GMLAS:` prefix. If this prefix it is omitted, then the GML driver is likely to be used.

Alternatively, starting with GDAL 3.10, specifying the `-if GMLAS` option to command line utilities accepting it, or `GMLAS` as the only value of the `papszAllowedDrivers` of *`GDALOpenEx()`*, also forces the driver to recognize the passed filename.

It is also possible to only used `GMLAS:` as the connection string, but in that case the schemas must be explicitly provided with the `XSD` open option.

7.27.3 Mapping of XML structure to OGR layers and fields

The driver scans the XML schemas referenced by the XML/GML to build the OGR layers and fields. It is strictly required that the schemas, directly or indirectly used, are fully valid. The content of the XML/GML file itself is marginally used, mostly to determine the SRS of geometry columns.

XML elements declared at the top level of a schema will generally be exposed as OGR layers. Their attributes and sub-elements of simple XML types (string, integer, real, ...) will be exposed as OGR fields. For sub-elements of complex type, different cases can happen. If the cardinality of the sub-element is at most one and it is not referenced by other elements, then it is "flattened" into its enclosing element. Otherwise it will be exposed as a OGR layer, with either a link to its "parent" layer if the sub-element is specific to its parent element, or through a junction table if the sub-element is shared by several parents.

By default the driver is robust to documents non strictly conforming to the schemas. Unexpected content in the document will be silently ignored, as well as content required by the schema and absent from the document.

Consult the *GMLAS mapping examples* page for more details.

By default in the configuration, swe:DataRecord and swe:DataArray elements from the Sensor Web Enablement (SWE) Common Data Model namespace will receive a special processing, so they are mapped more naturally to OGR concepts. The swe:field elements will be mapped as OGR fields, and the swe:values element of a swe:DataArray will be parsed into OGR features in a dedicated layer for each swe:DataArray. Note that those conveniency exposure is for read-only purpose. When using the write side of the driver, only the content of the general mapping mechanisms will be used.

7.27.4 Metadata layers

Three special layers "_ogr_fields_metadata", "_ogr_layers_metadata", "_ogr_layer_relationships" and "_ogr_other_metadata" add extra information to the basic ones you can get from the OGR data model on OGR layers and fields.

Those layers are exposed if the `EXPOSE_METADATA_LAYERS` open option is set to YES (or if enabled in the configuration). They can also be individually retrieved by specifying their name in calls to GetLayerByName(), or on as layer names with the ogrinfo and ogr2ogr utility.

Consult the *GMLAS metadata layers* page for more details.

7.27.5 Configuration file

A default configuration file gmlasconf.xml file is provided in the data directory of the GDAL installation. Its structure and content is documented in gmlasconf.xsd schema.

This configuration file enables the user to modify the following settings:

- whether remote schemas should be downloaded. Enabled by default.
- whether the local cache of schemas is enabled. Enabled by default.
- the path of the local cache. By default, $HOME/.gdal/gmlas_xsd_cache
- whether validation of the document against the schemas should be enabled. Disabled by default.
- whether validation error should cause dataset opening to fail. Disabled by default.
- whether the metadata layers should be exposed by default. Disabled by default.
- whether a 'ogr_pkid' field should always be generated. Disabled by default. Turning that on can be useful on layers that have a ID attribute whose uniqueness is not guaranteed among various documents. Which could cause issues when appending several documents into a target database table.
- whether layers and fields that are not used in the XML document should be removed. Disable by default.
- whether OGR array data types can be used. Enabled by default.

- whether the XML definition of the GML geometry should be reported as a OGR string field. Disabled by default.
- whether only XML elements that derive from gml:_Feature or gml:AbstractFeature should be considered in the initial pass of the schema building, when at least one element in the schemas derive from them. Enabled by default.
- several rules to configure if and how xlink:href should be resolved.
- a definition of XPaths of elements and attributes that must be ignored, so as to lighten the number of OGR layers and fields.

This file can be adapted and modified versions can be provided to the driver with the *CONFIG_FILE* open option. None of the elements of the configuration file are required. When they are absent, the default value indicated in the schema documentation is used.

Configuration can also be provided through other open options. Note that some open options have identical names to settings present in the configuration file. When such open option is provided, then its value will override the one of the configuration file (either the default one, or the one provided through the *CONFIG_FILE* open option).

7.27.6 Geometry support

XML schemas only indicate the geometry type but do not constraint the spatial reference systems (SRS), so it is theoretically possible to have object instances of the same class having different SRS for the same geometry field. This is not practical to deal with, so when geometry fields are detected, an initial scan of the document is done to find the first geometry of each geometry field that has an explicit srsName set. This one will be used for the whole geometry field. In case other geometries of the same field would have different SRS, they will be reprojected.

By default, only the OGR geometry built from the GML geometry is exposed in the OGR feature. It is possible to change the IncludeGeometryXML setting of the configuration file to true so as to expose a OGR string field with the XML definition of the GML geometry.

7.27.7 Performance issues with large multi-layer GML files.

Traditionally to read a OGR datasource, one iterate over layers with GDALDataset::GetLayer(), and for each layer one iterate over features with OGRLayer::GetNextFeature(). While this approach still works for the GMLAS driver, it may result in very poor performance on big documents or documents using complex schemas that are translated in many OGR layers.

It is thus recommended to use GDALDataset::GetNextFeature() to iterate over features as soon as they appear in the .gml/.xml file. This may return features from non-sequential layers, when the features include nested elements.

7.27.8 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are supported:

- **XSD=value:** to specify an explicit XSD application schema to use (or a list of filenames, provided they are comma separated). "http://" or "https://" URLs can be used. This option is not required when the XML/GML document has a schemaLocation attribute with valid links in its root element.
- **CONFIG_FILE=[<filename>/<xml>]:** filename of a XML configuration file conforming to the gmlasconf.xsd schema. It is also possible to provide the XML content directly inlined provided that the very first characters are <Configuration.
- **EXPOSE_METADATA_LAYERS=[YES/NO]:** Defaults to `NO`. whether the metadata layers "_ogr_fields_metadata", "_ogr_layers_metadata", "_ogr_layer_relationships" and "_ogr_other_metadata" should be reported by default.
- **VALIDATE=[YES/NO]:** Defaults to `NO`. whether the document should be validated against the schemas. Validation is done at dataset opening.

- **FAIL_IF_VALIDATION_ERROR=[YES/NO]:** Defaults to `NO`. Whether a validation error should cause dataset opening to fail. (only used if *`VALIDATE=YES`*)
- **REFRESH_CACHE=[YES/NO]:** Defaults to `NO`. Whether remote schemas and documents pointed by xlink:href links should be downloaded from the server even if already present in the local cache. If the cache is enabled, it will be refreshed with the newly downloaded resources.
- **SWAP_COORDINATES=[AUTO/YES/NO]:** Defaults to `AUTO`. Whether the order of the x/y or long/lat coordinates should be swapped. In AUTO mode, the driver will determine if swapping must be done from the srsName. If the srsName is urn:ogc:def:crs:EPSG::XXXX and that the order of coordinates in the EPSG database for this SRS is lat,long or northing,easting, then the driver will swap them to the GIS friendly order (long,lat or easting,northing). For other forms of SRS (such as EPSG:XXXX), GIS friendly order is assumed and thus no swapping is done. When SWAP_COORDINATES is set to YES, coordinates will be always swapped regarding the order they appear in the GML, and when it set to NO, they will be kept in the same order.
- **REMOVE_UNUSED_LAYERS=[YES/NO]:** Defaults to `NO`. Whether unused layers should be removed from the reported layers.
- **REMOVE_UNUSED_FIELDS=[YES/NO]:** Defaults to `NO`. Whether unused fields should be removed from the reported layers.
- **HANDLE_MULTIPLE_IMPORTS=[YES/NO]:** Defaults to `NO`. Whether multiple imports with the same namespace but different schema are allowed.
- **SCHEMA_FULL_CHECKING=[YES/NO]:** Defaults to `NO`. Whether to be pedantic with XSD checking or to be forgiving e.g. if the invalid part of the schema is not referenced in the main document.

7.27.9 Creation support

The GMLAS driver can write XML documents in a schema-driven way by converting a source dataset (contrary to most other drivers that have read support that implement the CreateLayer() and CreateFeature() interfaces). The typical workflow is to use the read side of the GMLAS driver to produce a SQLite/Spatialite/ PostGIS database, potentially modify the features imported and re-export this database as a new XML document.

The driver will identify in the source dataset "top-level" layers, and in those layers will find which features are not referenced by other top-level layers. As the creation of the output XML is schema-driver, the schemas need to be available. There are two possible ways:

- either the result of the processing of the schemas was stored as the 4 _ogr_* metadata tables in the source dataset by using the *`EXPOSE_METADATA_LAYERS=YES`* open option when converting the source .xml),
- or the schemas can be specified at creation time with the *`INPUT_XSD`* creation option.

By default, the driver will "wrap" the features inside a WFS 2.0 wfs:FeatureCollection / wfs:member element. It is also possible to ask the driver to create instead a custom wrapping .xsd file that declares the ogr_gmlas:FeatureCollection / ogr_gmlas:featureMember XML elements.

Note that while the file resulting from the export should be XML valid, there is no strong guarantee that it will validate against the additional constraints expressed in XML schema(s). This will depend on the content of the features (for example if converting from a GML file that is not conformant to the schemas, the output of the driver will generally be not validating)

If the input layers have geometries stored as GML content in a _xml suffixed field, then the driver will compare the OGR geometry built from that XML content with the OGR geometry stored in the dedicated geometry field of the feature. If both match, then the GML content stored in the _xml suffixed field will be used, such as to preserve particularities of the initial GML content. Otherwise GML will be exported from the OGR geometry.

To increase export performance on very large databases, creating attribute indexes on the fields pointed by the 'layer_pkid_name' attribute in '_ogr_layers_metadata' might help.

7.27.9.1 ogr2ogr behavior

When using ogr2ogr / GDALVectorTranslate() to convert to XML/GML from a source database, there are restrictions to the options that can be used. Only the following options of ogr2ogr are supported:

- dataset creation options (see below)
- layer names
- spatial filter through -spat option.
- attribute filter through -where option

The effect of spatial and attribute filtering will only apply on top-levels layers. Sub-features selected through joins will not be affected by those filters.

7.27.9.2 Dataset creation options

Dataset creation options can be specified in command-line tools using the syntax `-dsco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The supported dataset creation options are:

- **INPUT_XSD=value:** to specify an explicit XSD application schema to use (or a list of filenames, provided they are comma separated). "http://" or "https://" URLs can be used. This option is not required when the source dataset has a _ogr_other_metadata with schemas and locations filled.
- **CONFIG_FILE=[<filename>/<xml>]:** filename of a XML configuration file conforming to the gmlasconf.xsd schema. It is also possible to provide the XML content directly inlined provided that the very first characters are <Configuration>.
- **LAYERS=value:** Comma separated list of layers to export as top-level features. The special value "{SPATIAL_LAYERS}" can also be used to specify all layers that have geometries. When LAYERS is not specified, the driver will identify in the source dataset "top-level" layers, and in those layers will find which features are not referenced by other top-level layers.
- **SRSNAME_FORMAT=[SHORT/OGC_URN/OGC_URL (Only valid for GML 3 output)]:** Defaults to `OGC_URL`. If SHORT, then srsName will be in the form AUTHORITY_NAME:AUTHORITY_CODE If OGC_URN, then srsName will be in the form urn:ogc:def:crs:AUTHORITY_NAME::AUTHORITY_CODE If OGC_URL, then srsName will be in the form http://www.opengis.net/def/crs/AUTHORITY_NAME/0/AUTHORITY_CODE For OGC_URN and OGC_URL, in the case the SRS is a SRS without explicit AXIS order, but that the same SRS authority code imported with ImportFromEPSGA() should be treated as lat/long or northing/easting, then the function will take care of coordinate order swapping.
- **INDENT_SIZE=[0-8]:** Defaults to `2`. Number of spaces for each indentation level.
- **COMMENT=value:** Comment to add at top of generated XML file as a XML comment.
- **LINEFORMAT=[CRLF/LF]:** End-of-line sequence to use. Defaults to CRLF on Windows and LF on other platforms.
- **WRAPPING=[WFS2_FEATURECOLLECTION/GMLAS_FEATURECOLLECTION]:** Defaults to `WFS2_FEATURECOLLECTION`. Whether to wrap features in a wfs:FeatureCollection or in a ogr_gmlas:FeatureCollection.
- **TIMESTAMP=value:** User-specified XML dateTime value for timestamp to use in wfs:FeatureCollection attribute. If not specified, current date time is used. Only valid for *`WRAPPING=WFS2_FEATURECOLLECTION`*.
- **WFS20_SCHEMALOCATION=value:** Defaults to `http://schemas.opengis.net/wfs/2.0/wfs.xsd`. Path or URL to wfs.xsd. Only valid for *`WRAPPING=WFS2_FEATURECOLLECTION`*.
- **GENERATE_XSD=[YES/NO]:** Defaults to `YES`. Whether to generate a .xsd file that has the structure of the wrapping ogr_gmlas:FeatureCollection / ogr_gmlas:featureMember elements. Only valid for *`WRAPPING=GMLAS_FEATURECOLLECTION`*.

- **OUTPUT_XSD_FILENAME=value:** Wrapping .xsd filename. If not specified, same basename as output file with .xsd extension. Note that it is possible to use this option even if *GENERATE_XSD=NO*, so that the wrapping .xsd appear in the schemaLocation attribute of the .xml file. Only valid for *WRAPPING=GMLAS_FEATURECOLLECTION*.

7.27.10 Examples

7.27.10.1 Example 1: Listing content of a data file

```
ogrinfo -ro GMLAS:my.gml
```

7.27.10.2 Example 2: Converting to PostGIS

```
ogr2ogr -f PostgreSQL PG:'host=myserver dbname=warmerda' GMLAS:my.gml -nlt CONVERT_TO_
↪LINEAR
```

7.27.10.3 Example 3: Converting to Spatialite and back to GML

```
ogr2ogr -f SQLite tmp.sqlite GMLAS:in.gml -dsco SPATIALITE=YES -nlt CONVERT_TO_LINEAR -
↪oo EXPOSE_METADATA_LAYERS=YES
ogr2ogr -f GMLAS out.gml tmp.sqlite
```

7.27.11 See Also

- *GML*: general purpose driver not requiring the presence of schemas, but with limited support for complex features
- *NAS/ALKIS*: specialized GML driver for cadastral data in Germany

7.27.12 Credits

Initial implementation has been funded by the European Union's Earth observation programme Copernicus, as part of the tasks delegated to the European Environment Agency.

Development of special processing of some Sensor Web Enablement (SWE) Common Data Model swe:DataRecord and swe:DataArray constructs has been funded by Bureau des Recherches Géologiques et Minières (BRGM).

7.27.12.1 GMLAS - Mapping examples

This page gives a few examples of how XML constructs are mapped to OGR layers and fields by the *GMLAS - Geography Markup Language (GML) driven by application schemas* driver.

Table 2: Mapping examples

Schema	Document	OGR layers	Comments

`<schema xmlns="http://www.w3.org/2001/XMLSchema">`
`<element name="MyFeature">`
`<complexType>`
`  <sequence>`
`  <element name="name" type="string"/>`
`  </sequence>`
`  <attribute name="id" type="ID" use="required"/>`
`  <attribute name="attr" type="string"/>`
`</complexType>`
`</element>`
`</schema>`
	`<MyFeature id="my_id" attr="attr_value">`
	`  <name>my_name</name>`
	`</MyFeature>`
		`Layer name: MyFeature`
		`Geometry: None`
		`id: String (0.0) NOT NULL`
		`attr: String (0.0)`
		`name: String (0.0) NOT NULL`
		`OGRFeature(MyFeature)`
		`  id (String) = my_id`
		`  attr (String) = attr_value`
		`  name (String) = my_name`
			Element with attributes and sub-elements of simple type and maximum cardinality of 1.

`<schema xmlns="http://www.w3.org/2001/XMLSchema">`
`<element name="MyFeature">`
`<complexType>`
`  <sequence>`
`  <element name="str_array" type="string" maxOccurs="2"/>`
`  <element name="dt_array" type="dateTime" maxOccurs="unbounded"/>`
`  </sequence>`
`  <attribute name="id" type="ID" use="required"/>`
`</complexType>`
`</element>`
`</schema>`
	`<MyFeature id="my_id">`
	`  <str_array>first string</str_array>`
	`  <str_array>second string</str_array>`
	`  <dt_array>2016-09-24T15:31:00Z</dt_array>`
	`  <dt_array>2016-09-24T15:32:00Z</dt_array>`
	`</MyFeature>`
		`Layer name: MyFeature`
		`Geometry: None`
		`id: String (0.0) NOT NULL`
		`str_array: StringList (0.0) NOT NULL`
		`OGRFeature(MyFeature)`
		`  id (String) = my_id`
		`  str_array (StringList) = (2:first string,second string)`
		
		`Layer name: MyFeature_dt_array`
		`Geometry: None`
		`ogr_pkid: String (0.0) NOT NULL`
		`parent_id: String (0.0) NOT NULL`
		`value: DateTime (0.0)`
		`OGRFeature(MyFeature_dt_array):1`
		`  ogr_pkid (String) = my_id_dt_array_1`
		`  parent_id (String) = my_id`
			Example with array and child layer

swe:DataArray

The following snippet

```
<swe:DataArray>
    <swe:elementCount>
        <swe:Count>
                <swe:value>2</swe:value>
        </swe:Count>
    </swe:elementCount>
    <swe:elementType name="Components">
        <swe:DataRecord>
                <swe:field name="myTime">
                    <swe:Time definition="http://www.opengis.net/def/property/OGC/0/
↪SamplingTime">
                            <swe:uom xlink:href="http://www.opengis.net/def/uom/ISO-8601/
↪0/Gregorian"/>
                    </swe:Time>
                </swe:field>
                <swe:field name="myCategory">
                    <swe:Category definition="http://dd.eionet.europa.eu/vocabulary/aq/
↪observationverification"/>
                </swe:field>
                    <swe:field name="myQuantity">
                    <swe:Quantity definition="http://dd.eionet.europa.eu/vocabulary/aq/
↪primaryObservation/hour">
                            <swe:uom xlink:href="http://dd.eionet.europa.eu/vocabulary/
↪uom/concentration/ug.m-3"/>
                    </swe:Quantity>
                </swe:field>
                <swe:field name="myCount">
                    <swe:Count definition="http://"/>
                </swe:field>
                    <swe:field name="myText">
                    <swe:Text definition="http://"/>
                </swe:field>
                    <swe:field name="myBoolean">
                    <swe:Boolean definition="http://"/>
                </swe:field>
        </swe:DataRecord>
    </swe:elementType>
    <swe:encoding>
            <swe:TextEncoding decimalSeparator="." blockSeparator="@@" tokenSeparator=",
↪"/>
    </swe:encoding>
    <swe:values>2016-09-01T00:00:00+01:00,1,2.34,3,foo,true@@2017-09-01T00:00:00,2,3.45</
↪swe:values>
</swe:DataArray>
```

will receive a special processing to be mapped into a dedicated layer:

```
Layer name: dataarray_1_components
Geometry: None
Feature Count: 2
```

(continues on next page)

(continued from previous page)

```
Layer SRS WKT:
(unknown)
parent_ogr_pkid: String (0.0) NOT NULL
mytime: DateTime (0.0)
mycategory: String (0.0)
myquantity: Real (0.0)
mycount: Integer (0.0)
mytext: String (0.0)
myboolean: Integer(Boolean) (0.0)
OGRFeature(dataarray_1_components):1
  parent_ogr_pkid (String) = BAE8440FC4563A80D2AB1860A47AA0A3_DataArray_1
  mytime (DateTime) = 2016/09/01 00:00:00+01
  mycategory (String) = 1
  myquantity (Real) = 2.34
  mycount (Integer) = 3
  mytext (String) = foo
  myboolean (Integer(Boolean)) = 1

OGRFeature(dataarray_1_components):2
  parent_ogr_pkid (String) = BAE8440FC4563A80D2AB1860A47AA0A3_DataArray_1
  mytime (DateTime) = 2017/09/01 00:00:00
  mycategory (String) = 2
  myquantity (Real) = 3.45
```

See Also

- *main documentation page for GMLAS driver*

7.27.12.2 GMLAS - Metadata layers

This page details the structure of the extra metadata layers reported by the *GMLAS - Geography Markup Language (GML) driven by application schemas* driver.

_ogr_fields_metadata layer

This layer gives metadata about OGR fields, and also "abstract" fields that describe the relation ship between a parent and child layer.

Its fields are:

- `layer_name`: Name of the layer to which the field belongs to
- `field_name`: Name of the field. May be null when field_category is PATH_TO_CHILD_ELEMENT_NO_LINK or GROUP
- `field_xpath`: XPath of the element/attribute whose content is used for the field. The XPath is relative to the element that is the direct parent of this element/attribute, or a parent element in case of flattening. May be null for a field generated by OGR.
- `field_type`: the XML schema base data type (string, int, long, ID, ...). Extended with "geometry". May be null for a field generated by OGR
- `field_is_list`: Whether the XML type is a list.
- `field_min_occurs`: Integer value with the minimum number of occurrences of values. 0 or 1 typically. Or more for array types. May be null for a field generated by OGR

- `field_max_occurs`: Integer value with the maximum number of occurrences of values. 1 typically. Or more for array types. 2147483647 means unlimited. May be null for a field generated by OGR
- `field_repetition_on_sequence`: Boolean value to indicate if the field is related to a <sequence maxOccurs=">1 or unbounded"> construct. Only set when field_max_occurs is not 0 or 1.
- `field_default_value`: Default value of the field, or null
- `field_fixed_value`: Fixed value of the field, or null
- `field_category`: Category of the field. May be REGULAR, PATH_TO_CHILD_ELEMENT_NO_LINK, PATH_TO_CHILD_ELEMENT_WITH_LINK, PATH_TO_CHILD_ELEMENT_WITH_JUNCTION_TABLE, GROUP or SWE_FIELD. May be null for a field generated by OGR.
- `field_related_layer`: Name of the child layer for field_category != REGULAR
- `field_junction_layer`: Name of the junction layer. Only set if field_category is equal to PATH_TO_CHILD_ELEMENT_WITH_JUNCTION_TABLE
- `field_documentation`: Documentation from the schema.

Explanation of field_category values:

- REGULAR: the field is made from the value of an element or attribute that is a direct child of the element that is the root of the layer considered.
- PATH_TO_CHILD_ELEMENT_NO_LINK: A field declared with this category is not instantiated as a OGR field of the layer 'layer_name'. It is merely there to declare the relationship between the parent and child layers. This is when a sub-element is of a complex type or a repeated sub-element of simple type that does not match one of the OGR array types.
- PATH_TO_CHILD_ELEMENT_WITH_LINK: the content of this field is the primary key of a OGR feature of another layer. The field_related_layer field contains the name of that linked layer.
- PATH_TO_CHILD_ELEMENT_WITH_JUNCTION_TABLE: A field declared with this category is not instantiated as a OGR field of the layer 'layer_name'. It is merely there to declare the relationship between the parent and child layers. This is when the link between the parent and child layer is done through a junction table (case where the child layer is referenced by other parent layers).
- GROUP: A field declared with this category is not instantiated as a OGR field of the layer 'layer_name'. It is merely there to declare the relationship between the parent and child layers. This is when a layer uses XML schema group constructs with repeated cardinality.
- SWE_FIELD: A field derived from special processing of swe:DataRecord or swe:DataArray elements.

_ogr_layers_metadata layer

This layer gives metadata about OGR layers.

Its fields are:

- `layer_name`: Name of the layer
- `layer_xpath`: XPath of the element that is used as the root element for the layer. May be suffixed with ";extra=XXXX" for group constructs or repeated sequences of repeated elements, so as to distinguish for the XPath of their parent element. Will be null for junction tables or SWE_DATA_ARRAY layers.
- `layer_category`: Category of the layer. One of TOP_LEVEL_ELEMENT, NESTED_ELEMENT, JUNCTION_TABLE or SWE_DATA_ARRAY
- `layer_pkid_name`: Name of the primary key field. This is the text attribute that uniquely identified a feature (in its layer). This is the XML attribute/name of type xs:ID when it exists, otherwise a "ogr_pkid" field is automatically created. Will be null for SWE_DATA_ARRAY layers.

- `layer_parent_pkid_name`: Name of the field that is a foreign key to the parent layer of this layer. Only set for a NESTED_ELEMENT layer.
- `layer_documentation`: Documentation from the schema.

_ogr_layer_relationships layer

This layer gives metadata about relationship between OGR layers.

Its fields are:

- `parent_layer`: Name of the parent layer
- `parent_pkid`: Name of the primary key of the parent layer
- `parent_element_name`: Name of the XML element that links from the parent to the child. Will be null when the child layer is due to group constructs or repeated sequences of repeated elements of the parent.
- `child_layer`: Name of the child layer
- `child_pkid`: Name of the primary key of the child layer. Will be null for SWE_DATA_ARRAY layers

_ogr_other_metadata layer

This layer contains key / value pairs with different information.

Its fields are:

- `key`: Name of the metadata item
- `value`: Value of the metadata imtem

Possible keys are :

- document_filename: Filename of the XML/GML file read.
- configuration_filename: Filename of the XML configuration file used.
- configuration_inlined: XML content of the configuration file.
- namespace_uri_XX: URI of a namespace referenced by the schema(s).
- namespace_location_XX: Location of a schema.
- namespace_prefix_XX: Prefix of a namespace referenced by the schema(s).
- gml_version: GML version, such as 2.1.2, 3.1.1 or 3.2.1

See Also

- *main documentation page for GMLAS driver*

7.28 GMT ASCII Vectors (.gmt)

Driver short name

GMT

Driver built-in by default

This driver is built-in by default

OGR supports reading and writing GMT ASCII vector format. This is the format used by the Generic Mapping Tools (GMT) package, and includes recent additions to the format to handle more geometry types, and attributes. Currently GMT files are only supported if they have the extension ".gmt". Old (simple) GMT files are treated as either point, or linestring files depending on whether a ">" line is encountered before the first vertex. New style files have a variety of auxiliary information including geometry type, layer extents, coordinate system and attribute field declarations in comments in the header, and for each feature can have attributes.

7.28.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.28.2 Creation Issues

The driver supports creating new GMT files, and appending additional features to existing files, but update of existing features is not supported. Each layer is created as a separate .gmt file. If a name that ends with .gmt is not given, then the GMT driver will take the layer name and add the ".gmt" extension.

Writing to /dev/stdout or /vsistdout/ is supported since GDAL 3.5.0 (note that the file will then lack the optional region/bounding box header item)

7.29 GPKG -- GeoPackage vector

Driver short name

GPKG

Build dependencies

libsqlite3

This driver implements support for access to spatial tables in the OGC GeoPackage format standard The GeoPackage standard uses a SQLite database file as a generic container, and the standard defines:

- Expected metadata tables (`gpkg_contents`, `gpkg_spatial_ref_sys`, `gpkg_geometry_columns`)
- Binary format encoding for geometries in spatial tables (basically a GPKG standard header object followed by ISO standard well-known binary (WKB))
- Naming and conventions for extensions (extended feature types) and indexes (how to use SQLite r-tree in an interoperable manner)

This driver reads and writes SQLite files from the file system, so it must be run by a user with read/write access to the files it is working with.

The driver also supports reading and writing the following non-linear geometry types: CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE and MULTISURFACE

GeoPackage raster/tiles are supported. See the *GeoPackage raster* documentation page.

Validation whether a GeoPackage file conforms to the GeoPackage specification can be done with *gdal driver gpkg validate*.

7.29.1 Driver capabilities

Supports Create()

This driver supports the *`GDALDriver::Create()`* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.29.2 Specification version

GeoPackage version 1.0, 1.1, 1.2, 1.3 or 1.4 can be specified through the VERSION dataset creation option. Starting with GDAL 3.11, it defaults to 1.4. For earlier versions, it defaults to 1.2.

7.29.3 Limitations

- GeoPackage only supports one geometry column per table.

7.29.4 SQL

The driver supports OGR attribute filters, and users are expected to provide filters in the SQLite dialect, as they will be executed directly against the database.

SQL SELECT statements passed to ExecuteSQL() are also executed directly against the database.

Warning

Before GDAL 3.13, if the SQLite dialect is used with the SpatiaLite functions, an explicit cast operator AsGPB() is required to transform SpatiaLite geometries into GeoPackage geometries.

```
ogrinfo -update points.gpkg -sql "INSERT INTO points (geom) values (AsGPB(ST_
↪GeomFromText('POINT(3 54)', 4326)))"
```

Since GDAL 3.13, the use of AsGPB() is no longer needed (GDAL will automatically remove it when detecting `AsGPB(ST_` patterns)

The reverse conversion from GPKG geometries into SpatiaLite geometries is automatically done.

The "DROP TABLE layer_name" and "ALTER TABLE layer_name RENAME TO new_layer" statements can be used. They will update GeoPackage system tables.

The "HasSpatialIndex('table_name','geom_col_name')" statement can be used for checking if the table has spatial index on the named geometry column.

When dropping a table, or removing records from tables, the space they occupied is not immediately released and kept in the pool of file pages that SQLite may reuse later. If you need to shrink the file to its minimum size, you need to issue an explicit "VACUUM" SQL request. Note that this will result in a full rewrite of the file.

7.29.4.1 SQL functions

The following SQL functions, from the GeoPackage specification, are available:

- ST_MinX(geom *Geometry*): returns the minimum X coordinate of the geometry
- ST_MinY(geom *Geometry*): returns the minimum Y coordinate of the geometry
- ST_MaxX(geom *Geometry*): returns the maximum X coordinate of the geometry
- ST_MaxY(geom *Geometry*): returns the maximum Y coordinate of the geometry
- ST_IsEmpty(geom *Geometry*): returns 1 if the geometry is empty (but not null), e.g. a POINT EMPTY geometry
- ST_GeometryType(geom *Geometry*): returns the geometry type: 'POINT', 'LINESTRING', 'POLYGON', 'MULTIPOLYGON', 'MULTILINESTRING', 'MULTIPOINT', 'GEOMETRYCOLLECTION'
- ST_SRID(geom *Geometry*): returns the SRID of the geometry
- GPKG_IsAssignable(expected_geom_type *String*, actual_geom_type *String*) : mainly, needed for the 'Geometry Type Triggers Extension'

The following functions, with identical syntax and semantics as in Spatialite, are also available :

- CreateSpatialIndex(table_name *String*, geom_column_name *String*): creates a spatial index (RTree) on the specified table/geometry column
- DisableSpatialIndex(table_name *String*, geom_column_name *String*): drops an existing spatial index (RTree) on the specified table/geometry column
- ST_Area(geom *Geometry*): compute the area in square units of the geometry SRS.
- ST_Area(geom *Geometry*, use_ellipsoid *boolean*): (GDAL >= 3.9): compute the area in square meters, considering the geometry on the ellipsoid (use_ellipsoid must be set to true/1).
- ST_Length(geom *Geometry*): (GDAL >= 3.10): compute the area in units of the geometry SRS.
- ST_Length(geom *Geometry*, use_ellipsoid *boolean*): (GDAL >= 3.10): compute the area in meters, considering the geometry on the ellipsoid (use_ellipsoid must be set to true/1).
- SetSRID(geom *Geometry*, srs_id *Integer*): overrides the geometry' SRS ID, without reprojection.
- ST_Transform(geom *Geometry*, target_srs_id *Integer*): reproject the geometry to the SRS of specified srs_id. If no SRS with that given srs_id is not found in gpkg_spatial_ref_sys, starting with GDAL 3.2, it will be interpreted as a EPSG code.

- ST_EnvIntersects(geom *Geometry*, minx *Double*, miny *Double*, maxx *Double*, maxy *Double*): (GDAL >= 3.7) Returns 1 if the minimum bounding box of geom intersects the bounding box defined by (minx,miny)-(maxx,maxy), or 0 otherwise.
- ST_EnvIntersects(geom1 *Geometry*, geom2 *Geometry*): (GDAL >= 3.7) Returns 1 if the minimum bounding box of geom1 intersects the one of geom2, or 0 otherwise. (Note: this function, as all others, does not automatically uses spatial indices)

The raster SQL functions mentioned at *Raster SQL functions* are also available.

Since GDAL 3.13, `ST_Hilbert` is available. It encodes a (x, y) pair as the Hilbert curve index (32-bit unsigned integer), for a curve covering the given bounding box.

Four variants are available:

- `ST_Hilbert(x, y, min_x, min_y, max_x, max_y)`, where (x, y) is the point of interest, and (min_x, min_y, max_x, max_y) a bounding box that contains the point.
- `ST_Hilbert(x, y, layer_name)`, where (x, y) is the point of interest, and `layer_name` the name of a layer whose extent is used as the bounding box for the computation.
- `ST_Hilbert(geometry, min_x, min_y, max_x, max_y)`, where geometry is a geometry, and (min_x, min_y, max_x, max_y) a bounding box that contains the geometry. The center of the bounding box of the geometry is used as the point to encode.
- `ST_Hilbert(geometry, layer_name)`, where geometry is a geometry, and `layer_name` the name of a layer whose extent is used as the bounding box for the computation. The center of the bounding box of the geometry is used as the point to encode.

This is typically used to spatially sort a set of geometries. Note that the *gdal vector sort* program can also be used for that purpose.

```
gdal vector sql --sql="SELECT * FROM cities ORDER BY ST_Hilbert(geometry, 'cities')" \
                --input=cities.gpkg --output=sorted_cities.parquet
```

or

```
gdal vector sort --method=hilbert --input=cities.gpkg --output=sorted_cities.parquet
```

7.29.4.2 Link with Spatialite

If GDAL has been compiled against Spatialite, it is also possible to use Spatialite SQL functions. Transformation from GPKG geometry binary to Spatialite geometry binary encoding is done implicitly

```
ogrinfo poly.gpkg -sql "SELECT ST_Buffer(geom,5) FROM poly"
```

Note that due to the loose typing mechanism of SQLite, if a geometry expression returns a NULL value for the first row, this will generally cause OGR not to recognize the column as a geometry column. It might be then useful to sort the results by making sure that non-null geometries are returned first:

```
ogrinfo poly.gpkg -sql "SELECT * FROM (SELECT ST_Buffer(geom,5) AS geom, * FROM poly)␣
↪ORDER BY geom IS NULL ASC"
```

7.29.5 Transaction support

The driver implements transactions at the database level, per rfc-54

7.29.6 Relationships

Added in version 3.6.

Many-to-many relationship retrieval is supported, respecting the OGC GeoPackage Related Tables Extension. One-to-many relationships will also be reported for tables which utilize FOREIGN KEY constraints.

Relationship creation, deletion and updating is supported since GDAL 3.7. Relationships can only be updated to change their base or related table fields, or the relationship related table type. It is not permissible to change the base or related table itself, or the mapping table details. If this is desired then a new relationship should be created instead.

Note that when a many-to-many relationship is created in a GeoPackage, GDAL will always insert the mapping table into the gpkg_contents table. Formally this is not required by the Related Tables Extension (instead, the table should only be listed in gpkgext_relations), however failing to list the mapping table in gpkg_contents prevents it from being usable in some other applications (e.g. ESRI software).

7.29.7 Dataset open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are available:

- **LIST_ALL_TABLES=[AUTO/YES/NO]:** Defaults to `AUTO`. Whether all tables, including those not listed in gpkg_contents, should be listed. If AUTO, all tables including those not listed in gpkg_contents will be listed, except if the aspatial extension is found or a table is registered as 'attributes' in gpkg_contents. If YES, all tables including those not listed in gpkg_contents will be listed, in all cases. If NO, only tables registered as 'features', 'attributes' or 'aspatial' will be listed.
- **PRELUDE_STATEMENTS=value:** (GDAL >= 3.2) SQL statement(s) to send on the SQLite3 connection before any other ones. In case of several statements, they must be separated with the semicolon (;) sign. This option may be useful to attach another database to the current one and issue cross-database requests.

  > **Note**
  >
  > The attached database must be a GeoPackage one too, so that its geometry blobs are properly recognized (so typically not a Spatialite one)

- **NOLOCK=[YES/NO]:** (GDAL >= 3.4.2) Defaults to `NO`. Whether the database should be used without doing any file locking. Setting it to YES will only be honoured when opening in read-only mode and if the journal mode is not WAL. This corresponds to the nolock=1 query parameter described at https://www.sqlite.org/uri.html
- **IMMUTABLE=[YES/NO]:** (GDAL >= 3.5.3) Whether the database should be opened by assuming that the file cannot be modified by another process. This will skip any checks for change detection. This can be useful for WAL enabled files on read-only storage. GDAL will automatically try to turn it on when not being able to open in read-only mode a WAL enabled file. This corresponds to the immutable=1 query parameter described at https://www.sqlite.org/uri.html

Note: open options are typically specified with "-oo name=value" syntax in most OGR utilities, or with the `GDALOpenEx()` API call.

Note: configuration option *OGR_SQLITE_JOURNAL* can be used to set the journal mode of the GeoPackage (and thus SQLite) file, see also https://www.sqlite.org/pragma.html#pragma_journal_mode.

7.29.8 Creation issues

When creating a new GeoPackage file, the driver will attempt to force the database into a UTF-8 mode for text handling, satisfying the OGR strict UTF-8 capability. For pre-existing files, the driver will work with whatever it is given.

The driver updates the GeoPackage `last_change` timestamp when the file is created or modified. If consistent binary output is required for reproducibility, the timestamp can be forced to a specific value by setting the *OGR_CURRENT_DATE* global configuration option. When setting the option, take care to meet the specific time format requirement of the GeoPackage standard, e.g. for version 1.2.

7.29.8.1 Dataset creation options

Dataset creation options can be specified in command-line tools using the syntax `-dsco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following creation options (specific to vector, or common with raster) are available:

- **VERSION=[AUTO/1.0/1.1/1.2/1.3/1.4]:** Set GeoPackage version (for application_id and user_version fields). In AUTO mode, this will be equivalent to 1.4 starting with GDAL 3.11 (1.2 in prior versions) 1.3 is available starting with GDAL 3.3 1.4 is available starting with GDAL 3.7.1
- **ADD_GPKG_OGR_CONTENTS=[YES/NO]:** Defaults to `YES`. Defines whether to add a gpkg_ogr_contents table to keep feature count, and associated triggers.
- **DATETIME_FORMAT=[WITH_TZ/UTC]:** (GDAL >= 3.2) Defaults to `WITH_TZ`. Defines whether to keep the DateTime values in the time zones as used in the data source (WITH_TZ), or to convert the date and time expressions to UTC (Coordinated Universal Time). Pedantically, non-UTC time zones are not currently supported by GeoPackage v1.3 (see https://github.com/opengeospatial/geopackage/issues/530). When using UTC format, with a unspecified timezone, UTC will be assumed.
- **CRS_WKT_EXTENSION=[YES/NO]:** (GDAL >= 3.8) Defaults to `NO`. Defines whether to add the `definition_12_063` column to the `gpkg_spatial_ref_sys` system table, according to http://www.geopackage.org/spec/#extension_crs_wkt . The default is NO, unless the tile gridded coverage extension is used. With VERSION >= 1.4, a `epoch` column is also added. WKT strings in `definition_12_063` will follow the WKT2:2015 standard when possible, but may use the WKT2:2019 standard for specific cases (dynamic CRS with coordinate epoch). This option generally does not need to be specified, as the driver will automatically update the `gpkg_spatial_ref_sys` table when needed, but it may be useful to create GeoPackage datasets matching the exceptions of other software or profiles (such as the DGIWG-GPKG profile).
- **METADATA_TABLES=[YES/NO]:** (GDAL >= 3.8) Defines whether to add the metadata system tables. By default, they are created on demand. If NO is specified, they are not created, even if metadata is set. If YES is specified, they are always created.

Other options are available for raster. See the *GeoPackage raster* documentation page.

7.29.8.2 Layer creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALDatasetCreateLayer()* (C) or *Dataset.CreateLayer* (Python). The following layer creation options are available:

- **LAUNDER=[YES/NO]:** (GDAL >= 3.9) Defaults to `NO`. Whether layer and field names will be laundered. Laundering makes sure that the recommendation of https://www.geopackage.org/guidance/getting-started.html is followed: an identifier should start with a lowercase character and only use lowercase characters, numbers 0-9, and underscores (_). UTF-8 accented characters in the Latin-1 Supplement and Latin Extented-A sets are replaced when possible with the closest ASCII letter. Characters that do not match the recommendation are replaced with underscore. Consequently this option is not appropriate for non-Latin languages.
- **GEOMETRY_NAME=value:** Defaults to `geom`. Column to use for the geometry column. Default to "geom".
- **GEOMETRY_NULLABLE=[YES/NO]:** Defaults to `YES`. Whether the values of the geometry column can be NULL. Can be set to NO so that geometry is required.
- **SRID=<integer>:** (GDAL >= 3.9) Forced `srs_id` of the entry in the `gpkg_spatial_ref_sys` table to point to. This may be -1 ("Undefined Cartesian SRS"), 0 ("Undefined geographic SRS"), 99999 ("Undefined SRS"), a

valid EPSG CRS code or an existing entry of the `gpkg_spatial_ref_sys` table. If pointing to a non-existing entry, only a warning will be emitted.

- **DISCARD_COORD_LSB=[YES/NO]:** (GDAL >= 3.9) Defaults to `NO`. Whether the geometry coordinate precision should be used to set to zero non-significant least-significant bits of geometries. Helps when further compression is used. See *Geometry coordinate precision* for more details.
- **UNDO_DISCARD_COORD_LSB_ON_READING=[YES/NO]:** (GDAL >= 3.9) Defaults to `NO`. Whether to ask GDAL to take into coordinate precision to undo the effects of DISCARD_COORD_LSB. See *Geometry coordinate precision* for more details.
- **FID=value:** Defaults to `fid`. Column name to use for the OGR FID (primary key in the SQLite database).
- **OVERWRITE=[YES/NO]:** Defaults to `NO`. If set to "YES" will delete any existing layers that have the same name as the layer being created.
- **SPATIAL_INDEX=[YES/NO]:** Defaults to `YES`. If set to "YES" will create a spatial index for this layer.
- **PRECISION=[YES/NO]:** Defaults to `YES`. This may be "YES" to force new fields created on this layer to try and represent the width of text fields (in terms of UTF-8 characters, not bytes), if available using TEXT(width) types. If "NO" then the type TEXT will be used instead.
- **TRUNCATE_FIELDS=[YES/NO]:** Defaults to `NO`. This may be "YES" to force truncated of field values that exceed the maximum allowed width of text fields, and also to "fix" the passed string if needed to make it a valid UTF-8 string. If "NO" then the value is not truncated nor modified.
- **IDENTIFIER=value:** Identifier of the layer, as put in the contents table.
- **DESCRIPTION=value:** Description of the layer, as put in the contents table.
- **ASPATIAL_VARIANT=[GPKG_ATTRIBUTES/NOT_REGISTERED]:** Defaults to `GPKG_ATTRIBUTES`. How to register non spatial tables. Defaults to GPKG_ATTRIBUTES. Starting with GeoPackage 1.2, non spatial tables are part of the specification. They are recorded with data_type="attributes" in the gpkg_contents table. It is also possible to use the NOT_REGISTERED option, in which case the non spatial table is not registered at all in any GeoPackage system tables.
- **DATETIME_PRECISION=[AUTO/MILLISECOND/SECOND/MINUTE]:** (GDAL >= 3.8) Defaults to `AUTO`. Determines the level of detail for datetime fields. Starting with GeoPackage 1.4, three variants of datetime formats are supported: truncated at minute (`MINUTE`), truncated at second (`SECOND`) or including milliseconds (`MILLISECOND`). In `AUTO` mode and GeoPackage 1.4, milliseconds are included only if non-zero. Selecting modes `MINUTE` or `SECOND` will raise a warning with GeoPackage < 1.4.

7.29.9 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The following configuration options are available:

- **OGR_SQLITE_JOURNAL=value:** can be used to set the journal mode of the SQLite file, see also https://www.sqlite.org/pragma.html#pragma_journal_mode.
- **OGR_SQLITE_CACHE=value:** see *Performance hints*.
- **OGR_SQLITE_SYNCHRONOUS=value:** see *Performance hints*.
-

OGR_SQLITE_LOAD_EXTENSIONS=[<extension1,...,extensionN>/ENABLE_SQL_LOAD_EXTENSION]: (GDAL >= 3.5.0) Comma separated list of names of shared libraries containing extensions to load at database opening. If a file cannot be loaded directly, attempts are made to load with various operating-system specific extensions added. So for example, if "samplelib" cannot be loaded, then names like "samplelib.so" or "samplelib.dylib" or "samplelib.dll" might be tried also. The special value `ENABLE_SQL_LOAD_EXTENSION` can be used to enable the use of the SQL `load_extension()` function, which is normally disabled in standard builds of sqlite3. Loading extensions as a potential security impact if they are untrusted.

- **OGR_SQLITE_PRAGMA=value:** with this option any SQLite pragma can be specified. The syntax is `OGR_SQLITE_PRAGMA = "pragma_name=pragma_value[,pragma_name2=pragma_value2]*"`.
- **OGR_CURRENT_DATE=value:** the driver updates the GeoPackage `last_change` timestamp when the file is created or modified. If consistent binary output is required for reproducibility, the timestamp can be forced to a specific value by setting this global configuration option. When setting the option, take care to meet the specific time format requirement of the GeoPackage standard, e.g. for version 1.2.
- **SQLITE_USE_OGR_VFS=[YES/NO]:** YES enables extra buffering/caching by the GDAL/OGR I/O layer and can speed up I/O. More information *here*. Be aware that no file locking will occur if this option is activated, so concurrent edits may lead to database corruption.
- **OGR_GPKG_NUM_THREADS=value:** (GDAL >= 3.8.3) Can be set to an integer or `ALL_CPUS`. This is the number of threads used when reading tables through the ArrowArray interface, when no filter is applied and when features have consecutive feature ID numbering. The default is the minimum of 4 and the number of CPUs. Note that setting this value too high is not recommended: a value of 4 is close to the optimal.

7.29.10 Metadata

GDAL uses the standardized gpkg_metadata and gpkg_metadata_reference tables to read and write metadata, on the dataset and layer objects.

GDAL metadata, from the default metadata domain and possibly other metadata domains, is serialized in a single XML document, conformant with the format used in GDAL PAM (Persistent Auxiliary Metadata) .aux.xml files, and registered with md_scope=dataset and md_standard_uri=http://gdal.org in gpkg_metadata. For the dataset, this entry is referenced in gpkg_metadata_reference with a reference_scope=geopackage. For a layer, this entry is referenced in gpkg_metadata_reference with a reference_scope=table and table_name={name of the table}

Metadata not originating from GDAL can be read by the driver and will be exposed as metadata items with keys of the form GPKG_METADATA_ITEM_XXX and values the content of the *metadata* columns of the gpkg_metadata table. Update of such metadata is not currently supported through GDAL interfaces (although it can be through direct SQL commands).

The specific DESCRIPTION and IDENTIFIER metadata item of the default metadata domain can be used in read/write to read from/update the corresponding columns of the gpkg_contents table.

7.29.11 Non-spatial tables

The core GeoPackage specification of GeoPackage 1.0 and 1.1 did not support non-spatial tables. This was added in GeoPackage 1.2 as the "attributes" data type.

The driver allows creating and reading non-spatial tables with the *GeoPackage aspatial extension*.

The driver will also, by default, list non spatial tables that are not registered through the gdal_aspatial extension, and support the GeoPackage 1.2 "attributes" data type as well. Non spatial tables are by default created following the GeoPackage 1.2 "attributes" data type (can be controlled with the *`ASPATIAL_VARIANT`* layer creation option).

7.29.12 Spatial views

Views can be created and recognized as valid spatial layers if a corresponding record is inserted into the gpkg_contents and gpkg_geometry_columns table.

In the case of the columns in the SELECT clause of the view acts a integer primary key, then it can be recognized by OGR as the FID column of the view, provided it is renamed as OGC_FID. Selecting a feature id from a source table without renaming will not be sufficient, since due to joins this feature id could appear several times. Thus the user must explicitly acknowledge that the column is really a primary key.

For example:

```
CREATE VIEW my_view AS SELECT foo.fid AS OGC_FID, foo.geom FROM foo JOIN another_table␣
↪ON foo.some_id = another_table.other_id;
INSERT INTO gpkg_contents (table_name, identifier, data_type, srs_id) VALUES ( 'my_view',
↪ 'my_view', 'features', 4326);
INSERT INTO gpkg_geometry_columns (table_name, column_name, geometry_type_name, srs_id,␣
↪z, m) values ('my_view', 'my_geom', 'GEOMETRY', 4326, 0, 0);
```

This requires GDAL to be compiled with the SQLITE_HAS_COLUMN_METADATA option and SQLite3 with the SQLITE_ENABLE_COLUMN_METADATA option. This can be easily verified if the SQLITE_HAS_COLUMN_METADATA=YES driver metadata item is declared (for example with "ogrinfo --format GPKG").

Starting with GDAL 3.7.1, it is possible to define a geometry column as the result of a Spatialite spatial function. Note however that this is an extension likely to be non-interoperable with other software that does not activate Spatialite for the SQLite3 database connection. Such geometry column should be registered into the `gpkg_extensions` using the `gdal_spatialite_computed_geom_column` extension name (cf *GeoPackage Spatialite computed geometry column extension*), like below:

```
CREATE VIEW my_view AS SELECT foo.fid AS OGC_FID, AsGBP(ST_Multi(foo.geom)) FROM foo;
INSERT INTO gpkg_contents (table_name, identifier, data_type, srs_id) VALUES (
    'my_view', 'my_view', 'features', 4326);
INSERT INTO gpkg_geometry_columns (table_name, column_name, geometry_type_name, srs_id,␣
↪z, m) VALUES (
    'my_view', 'my_geom', 'MULTIPOLYGON', 4326, 0, 0);
INSERT INTO gpkg_extensions (table_name, column_name, extension_name, definition, scope)␣
↪VALUES (
    'my_view', 'my_geom', 'gdal_spatialite_computed_geom_column',
    'https://gdal.org/drivers/vector/gpkg_spatialite_computed_geom_column.html', 'read-
↪write');
```

7.29.13 Coordinate Reference Systems

Valid geographic, projected and compound CRS supported in general by GDAL are also supported by GeoPackage and stored in the `gpkg_spatial_ref_sys` table.

Two special hard-coded CRS are reserved per the GeoPackage specification:

- srs_id=0, for a Undefined Geographic CRS. For GDAL 3.8 or earlier, this one is selected by default if creating a spatial layer without any explicit CRS
- srs_id=-1, for a Undefined Projected CRS. It might be selected by creating a layer with a CRS instantiated from the following WKT string: `LOCAL_CS["Undefined Cartesian SRS"]`. (GDAL >= 3.3)

Starting with GDAL 3.9, a layer without any explicit CRS is mapped from/to a custom entry of srs_id=99999 with the following properties:

- `srs_name`: `Undefined SRS`
- `organization`: `GDAL`
- `organization_coordsys_id`: 99999
- `definition`: `LOCAL_CS["Undefined SRS",LOCAL_DATUM["unknown",32767],UNIT["unknown",0],AXIS["Easting",EAST],AXIS["Northing",NORTH]]`
- `definition_12_063` (when the CRS WKT extension is used): `ENGCRS["Undefined SRS",EDATUM["unknown"],CS[Cartesian,2],AXIS["easting",east,ORDER[1],LENGTHUNIT["unknown",0]],AXIS["northing",north,ORDER[2],LENGTHUNIT["unknown",0]]]`
- `description`: `Custom undefined coordinate reference system`

Note that the use of a LOCAL_CS / EngineeringCRS is mostly to provide a valid CRS definition to comply with the requirements of the GeoPackage specification and to be compatible with other applications (or GDAL 3.8 or earlier), but the semantics of that entry is intended to be "undefined SRS of any kind".

7.29.14 Level of support of GeoPackage Extensions

(Restricted to those that have a vector scope)

Table 3: Extensions

Extension name	OGC adopted extension ?	Supported by GDAL?
Non-Linear Geometry Types	Yes	Yes
RTree Spatial Indexes	Yes	Yes
Metadata	Yes	Yes
Schema	Yes	Yes, since GDAL 3.3 (Geopackage constraints exposed as field domains)
WKT for Coordinate Reference Systems (WKT v2)	Yes	Yes
GeoPackage aspatial extension	No	Yes. Deprecated, in favor of the *attributes* official data_type
GeoPackage Spatialite computed geometry column extension	No	Yes, starting with GDAL 3.7.1
OGC GeoPackage Related Tables Extension	Yes	Yes, starting with GDAL 3.6

7.29.15 Compressed files

Starting with GDAL 3.7, the driver can also support reading and creating .gpkg.zip files containing one .gpkg file.

On large files, good read performance can only be achieved if the file inside the .zip is not compressed or compressed using the SOZip optimization.

Update of an existing file is not supported.

Creation involves the creation of a temporary file. Sufficiently large files will be automatically compressed using the SOZip optimization.

7.29.16 Geometry coordinate precision

Added in version GDAL: 3.9

The GeoPackage driver supports reading and writing the geometry coordinate precision, using the *OGRGeomCoordinatePrecision* settings of the *OGRGeomFieldDefn*. By default, the geometry coordinate precision is only noted in metadata, and does not cause geometries that are written to be modified to comply with this precision.

Several settings may be combined to apply further processing:

- if the *OGR_APPLY_GEOM_SET_PRECISION* configuration option is set to YES, the *OGRGeometry::SetPrecision()* method will be applied when calling the CreateFeature() and SetFeature() method of the driver, to round X and Y coordinates to the specified precision, and fix potential geometry invalidities resulting from the rounding.
- if the DISCARD_COORD_LSB layer creation option is set to YES, the less-significant bits of the WKB geometry encoding which are not relevant for the requested precision are set to zero. This can improve further lossless compression stages, for example when putting a GeoPackage in an archive. Note however that when reading back such geometries and displaying them to the maximum precision, they will not "exactly" match the original *OGRGeomCoordinatePrecision* settings. However, they will round back to it. The value of the DISCARD_COORD_LSB layer creation option is written in the dataset metadata, and will be re-used for later edition sessions.
- if the UNDO_DISCARD_COORD_LSB_ON_READING layer creation option is set to YES (only makes sense if the DISCARD_COORD_LSB layer creation option is also set to YES), when *reading* back geometries from a dataset, the *OGRGeometry::roundCoordinates()* method will be applied so that the geometry coordinates exactly match the original specified coordinate precision. That option will only be honored by GDAL 3.9 or later.

Implementation details: the coordinate precision is stored in a record in each of the gpkg_metadata and gpkg_metadata_reference table, with the following additional constraints on top of the ones imposed by the GeoPackage specification:

- gpkg_metadata.md_standard_uri = 'http://gdal.org'
- gpkg_metadata.mime_type = 'text/xml'
- gpkg_metadata.metadata = '<CoordinatePrecision xy_resolution="{xy_resolution}" z_resolution="{z_resolution}" m_resolution="{m_resolution}" discard_coord_lsb={true or false} undo_discard_coord_lsb_on_reading={true or false} />'
- gpkg_metadata_reference.reference_scope = 'column'
- gpkg_metadata_reference.table_name = '{table_name}'
- gpkg_metadata_reference.column_name = '{geometry_column_name}'

Note that the xy_resolution, z_resolution or m_resolution attributes of the XML CoordinatePrecision element are optional. Their numeric value is expressed in the units of the SRS for xy_resolution and z_resolution.

7.29.17 Performance hints

The same performance hints apply as those mentioned for the *SQLite driver*.

7.29.18 Examples

7.29.18.1 Example 1: Simple translation of a single shapefile into GeoPackage

The table 'abc' will be created with the features from abc.shp and attributes from abc.dbf. The file `filename.gpkg` must **not** already exist, as it will be created. For adding new layers into existing geopackage run ogr2ogr with **-update**.

```
ogr2ogr -f GPKG filename.gpkg abc.shp
```

7.29.18.2 Example 2: Updating an existing GeoPackage file

Updates an existing file – e.g. a GeoPackage template – by adding features to it from another GeoPackage file containing features according to the same or a backwards compatible database schema.

```
ogr2ogr -append output.gpkg input.gpkg
```

7.29.18.3 Example 3: Converting a directory of shapefiles into a GeoPackage

Each file will end up as a new table within the GPKG file. The file `filename.gpkg` must **not** already exist, as it will be created.

```
ogr2ogr -f GPKG filename.gpkg ./path/to/dir
```

7.29.18.4 Example 4: Converting a PostGIS database into a GeoPackage

Each table in the database will end up as a table in the GPKG file. The file `filename.gpkg` must **not** already exist, as it will be created.

```
ogr2ogr -f GPKG filename.gpkg PG:'dbname=mydatabase host=localhost'
```

7.29.18.5 Example 5: Performing a join between two GeoPackage databases

```
ogrinfo my_spatial.gpkg \
  -sql "SELECT poly.id, other.foo FROM poly JOIN other_schema.other USING (id)
↪" \
  -oo PRELUDE_STATEMENTS="ATTACH DATABASE 'other.gpkg' AS other_schema"
```

7.29.19 Secure deletion

Depending on how SQLite3 is built, secure deletion might or might not be enabled. Starting with GDAL 3.10, secure deletion is always enabled, unless `SECURE_DELETE` is specified through the *OGR_SQLITE_PRAGMA* configuration option. Note that secure deletion does not recover potential lost space, so running a VACUUM query is recommended to fully optimized a database that has been subject to updates or deletions.

7.29.20 See Also

- *GeoPackage raster* documentation page
- Getting Started With GeoPackage
- OGC GeoPackage format standard specification, HTML format (current/development version of the standard)
- OGC GeoPackage Encoding Standard page
- SQLite

7.29.20.1 GeoPackage aspatial extension

> **Warning**
>
> This extension has been superseded by GeoPackage 1.2 `attributes` data type.

GeoPackage 1.0 Extension

Extension follows template from Annex I of the OGC GeoPackage 1.0 Specification.

Extension Title

Aspatial Support

Introduction

Support for aspatial data (i.e. SQLite tables/views without a geometry column), potentially with associated metadata.

> **Warning**
>
> Deprecated in favor of the GeoPackage 'attributes' data type.

Extension Author

GDAL - Geospatial Data Abstraction Library, author_name *gdal*.

Extension Name or Template

SQL

```
INSERT INTO gpkg_extensions
(table_name, column_name, extension_name, definition, scope)
VALUES
(
    NULL,
    NULL,
    'gdal_aspatial',
    'http://gdal.org/geopackage_aspatial.html',
    'read-write'
);
```

Extension Type

Extension of Existing Requirement in Clause 2.

Applicability

This extension applies to any aspatial user data table or view specified in the `gpkg_contents` table with a lowercase *data_type* column value of "aspatial".

Scope

Read-write

Requirements

GeoPackage

Contents Table - Aspatial

The *gpkg_contents* table SHALL contain a row with a lowercase *data_type* column value of "aspatial" for each aspatial user data table or view.

User Data Tables

The second component of the SQL schema for aspatial tables in an Extended GeoPackage described in clause 'Contents Table - Aspatial' above are user tables or views that contain aspatial user data.

An Extended GeoPackage with aspatial support is not required to contain any user data tables. User data tables MAY be empty.

An Extended GeoPackage with aspatial support MAY contain tables or views. Every such aspatial table or view MAY have a column with column type INTEGER and PRIMARY KEY AUTOINCREMENT column constraints per EXAMPLE.

Column Name	Type	Description	Null	Default	Key
id	INTEGER	Autoincrement primary key	no		PK
text_attribute	TEXT	Text attribute of row	yes		
real_attribute	REAL	Real attribute of row	yes		
boolean_attribute	BOOLEAN	Boolean attribute of row	yes		
raster_or_photo	BLOB	Photograph	yes		

An integer primary key of an aspatial table or view allows features to be linked to row level metadata records in the *gpkg_metadata* table by SQLite ROWID values in the *gpkg_metadata_reference* table as described in clause 2.4.3 Metadata Reference Table.

An aspatial table or view SHALL NOT have a geometry column.

Columns in aspatial tables or views SHALL be defined using only the data types specified in Table 1 in Clause 1.1.1.1.3.

GeoPackage SQLite Configuration

None

GeoPackage SQLite Extension

None

7.29.20.2 GeoPackage Spatialite computed geometry column extension

GeoPackage 1.3.1 Extension

Extension follows template from Annex E of the OGC GeoPackage 1.3.1 Specification.

Extension Title

Spatialite computed geometry column

Introduction

Support for geometry column computed from the result of a SQL function of the Spatialite library.

Extension Author

GDAL - Geospatial Data Abstraction Library, author_name *gdal*.

Extension Name or Template

The extension name is `gdal_spatialite_computed_geom_column`.

Each view using the extension should register its geometry column with the following template:

```
INSERT INTO gpkg_extensions
(table_name, column_name, extension_name, definition, scope)
VALUES
(
    '{view_name}',
    '{geometry_column_name}',
    'gdal_spatialite_computed_geom_column',
    'https://gdal.org/drivers/vector/gpkg_spatialite_computed_geom_column.html',
    'read-write'
);
```

Extension Type

Extension of Existing Requirement in Clause 2.1 "Features".

Applicability

This extension applies to any view specified in the `gpkg_contents` table with a lowercase *data_type* column value of "features" which defines its geometry column as the result of a SQL function of the Spatialite library.

Scope

Read-write

Requirements

GeoPackage

A view using this extension should have its geometry column defined as the result of a Spatialite SQL function returning a geometry, and wrapped with the `AsGPB` function to return a GeoPackage geometry blob.

An example of such a view is:

```
CREATE VIEW my_view AS SELECT foo.fid AS OGC_FID, AsGBP(ST_Multi(foo.geom)) FROM foo
```

The view name and geometry column name should be registered in the `gpkg_extensions` table, as shown in the above Extension Template.

GeoPackage SQLite Configuration

None

GeoPackage SQLite Extension

This extension assumes that the Spatialite SQL functions are available at runtime.

7.30 GPSBabel

Driver short name

GPSBabel

Build dependencies

(read support needs GPX driver and libexpat)

The GPSBabel driver for now that relies on the GPSBabel utility to access various GPS file formats.

The GPSBabel executable must be accessible through the PATH.

7.30.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

7.30.2 Read support

The driver needs the *GPX* driver to be fully configured with read support (through Expat library) to be able to parse the output of GPSBabel, as GPX is used as the intermediate pivot format.

The returned layers can be waypoints, routes, route_points, tracks, track_points depending on the input data.

The syntax to specify an input datasource is : *GPSBabel:gpsbabel_file_format[,gpsbabel_format_option]*:[features=[waypoints,][tracks,]
where :

- *gpsbabel_file_format* is one of the file formats handled by GPSBabel.
- *gpsbabel_format_option* is any option handled by the specified GPSBabel format (refer to the documentation of each GPSBabel format)
- *features=* can be used to modify the type of features that GPSBabel will import. waypoints matches the -w option of gpsbabel commandline, tracks matches -t and routes matches -r. This option can be used to require full data import from GPS receivers that are slow and for which GPSBabel would only fetch waypoints by default. See the documentation on Route and Track modes for more details.
- *filename* can be an actual on-disk file, a file handled through the GDAL virtual file API, or a special device handled by GPSBabel such as "usb:", "/dev/ttyS0", "COM1:", etc.. What is actually supported depends on the used GPSBabel format.

Alternatively, for a few selected GPSBabel formats, just specifying the filename might be sufficient. The list includes for now :

- garmin_txt
- gtrnctr
- gdb
- magellan
- mapsend
- mapsource
- nmea
- osm
- ozi
- igc

The *USE_TEMPFILE=YES* configuration option can be used to create an on-disk temporary GPX file instead of a in-memory one, when reading big amount of data.

7.30.3 Write support

The driver relies on the GPX driver to create an intermediate file that will be finally translated by GPSBabel to the desired GPSBabel format. (The GPX driver does not need to be configured for read support for GPSBabel write support.).

The support geometries, options and other creation issues are the ones of the GPX driver. Please refer to its *documentation* for more details.

The syntax to specify an output datasource is : *GPSBabel:gpsbabel_file_format[,gpsbabel_format_option]*:filename* where :

- *gpsbabel_file_format* is one of the file formats handled by GPSBabel.
- *gpsbabel_format_option* is any option handled by the specified GPSBabel format (refer to the documentation of each GPSBabel format)

Alternatively, you can just pass a filename as output datasource name and specify the dataset creation option GPSBABEL_DRIVER=gpsbabel_file_format[,gpsbabel_format_option]*

When writing large datasets, the *USE_TEMPFILE* configuration option may be used:

- **USE_TEMPFILE=[YES/NO]:** Defaults to `NO`. If `YES`, create an on-disk temporary GPX file instead of a in-memory one, when writing big amount of data.

7.30.3.1 Examples

Example 1: Reading the waypoints from a Garmin USB receiver

```
ogrinfo -ro -al GPSBabel:garmin:usb:
```

Example 2: Converting a shapefile to Magellan Mapsend format

```
ogr2ogr -f GPSBabel GPSBabel:mapsend:out.mapsend in.shp
```

7.30.3.2 See Also

- GPSBabel Home Page
- GPSBabel file formats
- *GPX driver page*

7.31 GPX - GPS Exchange Format

Driver short name

GPX

Build dependencies

(read support needs libexpat)

GPX (the GPS Exchange Format) is a light-weight XML data format for the interchange of GPS data (waypoints, routes, and tracks) between applications and Web services on the Internet.

OGR has support for GPX reading (if GDAL is build with *expat* library support) and writing.

Version supported are GPX 1.0 and 1.1 for reading, GPX 1.1 for writing.

The OGR driver supports reading and writing of all the GPX feature types:

- *waypoints* : layer of features of OGR type wkbPoint
- *routes* : layer of features of OGR type wkbLineString
- *tracks* : layer of features of OGR type wkbMultiLineString

It also supports reading of route points and track points in standalone layers (*route_points* and *track_points*), so that their own attributes can be used by OGR.

In addition to its GPX attributes, each route point of a route has a *route_fid* (foreign key to the FID of its belonging route) and a *route_point_id* which is its sequence number in the route.
The same applies for track points with *track_fid*, *track_seg_id* and *track_seg_point_id*. All coordinates are relative to the WGS84 datum (EPSG:4326).

The OGR GPX driver reads and writes the GPX attributes for the waypoints, routes and tracks.

Starting with GDAL 3.7, metadata from the top <metadata> element is reported at dataset level.

7.31.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.31.2 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The following configuration options are available:

- **GPX_USE_EXTENSIONS=value:** See *Extensions element reading*.
- **GPX_SHORT_NAMES=[TRUE/FALSE]:** Defaults to `FALSE`. If `TRUE`, the following fields will be renamed:
  - "track_seg_id" to "trksegid"
  - "track_seg_point_id" to "trksegptid"
  - "route_point_id" to "rteptid"

  But note that no particular processing will be done for any extension field names.
- **GPX_ELE_AS_25D=[YES/NO]:** Defaults to `NO`. If `YES`, the elevation element will be used to set the Z coordinates of waypoints, route points and track points.
- **GPX_N_MAX_LINKS=value:** Defaults to `2`. Determines the number of *<link>* elements can be taken into account by feature.

7.31.3 Open options

Added in version 3.9.

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are available:

- **SHORT_NAMES=[YES/NO]:** Defaults to `FALSE`. If `YES`, the following fields will be renamed:
  - "track_seg_id" to "trksegid"
  - "track_seg_point_id" to "trksegptid"
  - "route_point_id" to "rteptid"

  But note that no particular processing will be done for any extension field names.
- **ELE_AS_25D=[YES/NO]:** Defaults to `NO`. If `YES`, the elevation element will be used to set the Z coordinates of waypoints, route points and track points.
- **N_MAX_LINKS=value:** Defaults to `2`. Determines the number of *<link>* elements can be taken into account by feature.

7.31.4 Encoding issues

Expat library supports reading the following built-in encodings :

- US-ASCII
- UTF-8
- UTF-16
- ISO-8859-1
- Windows-1252

The content returned by OGR will be encoded in UTF-8, after the conversion from the encoding mentioned in the file header is.

If your GPX file is not encoded in one of the previous encodings, it will not be parsed by the GPX driver. You may convert it into one of the supported encoding with the *iconv* utility for example and change accordingly the *encoding* parameter value in the XML header.

When writing a GPX file, the driver expects UTF-8 content to be passed in.

7.31.5 Extensions element reading

If the <*extensions*> element is detected in a GPX file, OGR will expose the content of its sub elements as fields. Complex content of sub elements will be exposed as an XML blob.

The following sequence GPX content :

```
<extensions>
    <navaid:name>TOTAL RF</navaid:name>
    <navaid:address>BENSALEM</navaid:address>
    <navaid:state>PA</navaid:state>
    <navaid:country>US</navaid:country>
    <navaid:frequencies>
    <navaid:frequency type="CTAF" frequency="122.900" name="CTAF"/>
    </navaid:frequencies>
    <navaid:runways>
    <navaid:runway designation="H1" length="80" width="80" surface="ASPH-G">
    </navaid:runway>
    </navaid:runways>
    <navaid:magvar>12</navaid:magvar>
</extensions>
```

will be interpreted in the OGR SF model as :

```
navaid_name (String) = TOTAL RF
navaid_address (String) = BENSALEM
navaid_state (String) = PA
navaid_country (String) = US
navaid_frequencies (String) = <navaid:frequency type="CTAF" frequency="122.900" name=
↪"CTAF" ></navaid:frequency>
navaid_runways (String) = <navaid:runway designation="H1" length="80" width="80" surface=
↪"ASPH-G" ></navaid:runway>
navaid_magvar (Integer) = 12
```

Note : the GPX driver will output content of the extensions element only if it is found in the first records of the GPX file. If extensions appear later, you can force an explicit parsing of the whole file with the *GPX_USE_EXTENSIONS* configuration option.

7.31.6 Creation Issues

On export all layers are written to a single GPX file. Update of existing files is not currently supported.

If the output file already exits, the writing will not occur. You have to delete the existing file first.

Supported geometries :

- Features of type wkbPoint/wkbPoint25D are written in the *wpt* element.
- Features of type wkbLineString/wkbLineString25D are written in the *rte* element.
- Features of type wkbMultiLineString/wkbMultiLineString25D are written in the *trk* element.
- Other type of geometries are not supported.

For route points and tracks points, if there is a Z coordinate, it is used to fill the elevation element of the corresponding points.

If a layer is named "track_points" with wkbPoint/wkbPoint25D geometries, the tracks in the GPX file will be built from the sequence of features in that layer. This is the way of setting GPX attributes for each track point, in addition to the raw coordinates. Points belonging to the same track are identified thanks to the same value of the 'track_fid' field (and it will be broken into track segments according to the value of the 'track_seg_id' field). They must be written in sequence so that track objects are properly reconstructed. The 'track_name' field can be set on the first track point to fill the <name> element of the track. Similarly, if a layer is named "route_points" with wkbPoint/wkbPoint25D geometries, the routes in the GPX file will be built from the sequence of points with the same value of the 'route_fid' field. The 'route_name' field can be set on the first track point to fill the <name> element of the route.

7.31.7 Layer creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALDatasetCreateLayer()* (C) or *Dataset.CreateLayer* (Python). The following layer creation options are supported:

- **FORCE_GPX_TRACK=[YES/NO]:** Defaults to `NO`. By default when writing a layer whose features are of type wkbLineString, the GPX driver chooses to write them as routes. If YES is specified, they will be written as tracks.
- **FORCE_GPX_ROUTE=[YES/NO]:** Defaults to `NO`. By default when writing a layer whose features are of type wkbMultiLineString, the GPX driver chooses to write them as tracks. If YES is specified, they will be written as routes, provided that the multilines are composed of only one single line.

7.31.8 Dataset creation options

Dataset creation options can be specified in command-line tools using the syntax `-dsco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following dataset creation options are supported:

- **GPX_USE_EXTENSIONS=[YES/NO]:** Defaults to `NO`. By default, the GPX driver will discard attribute fields that do not match the GPX XML definition (name, cmt, etc...). If *GPX_USE_EXTENSIONS=YES* is specified, extra fields will be written inside the*<extensions>* tag.
- **GPX_EXTENSIONS_NS=value:** Defaults to `ogr`. The namespace value used for extension tags. Only used if *GPX_USE_EXTENSIONS=YES* and *GPX_EXTENSIONS_NS_URL* is set.

- **GPX_EXTENSIONS_NS_URL=value:** Defaults to "http://osgeo.org/gdal". The namespace URI. Only used if *GPX_USE_EXTENSIONS=YES* and *GPX_EXTENSIONS_NS* is set.
- **LINEFORMAT=[CRLF/LF]:** By default files are created with the line termination conventions of the local platform (CR/LF on win32 or LF on all other systems). This may be overridden through use of this option.
- **CREATOR=value:** (GDAL >= 3.8) Name of creating application. Defaults to GDAL followed by its version number.

Dataset creation options to fill the top <metadata> element have been added in GDAL 3.7:

- **METADATA_AUTHOR_EMAIL=value:** (GDAL >= 3.7)
- **METADATA_AUTHOR_NAME=value:** (GDAL >= 3.7)
- **METADATA_AUTHOR_LINK_HREF=value:** (GDAL >= 3.7)
- **METADATA_AUTHOR_LINK_TEXT=value:** (GDAL >= 3.7)
- **METADATA_AUTHOR_LINK_TYPE=value:** (GDAL >= 3.7)
- **METADATA_COPYRIGHT_AUTHOR=value:** (GDAL >= 3.7)
- **METADATA_COPYRIGHT_LICENSE=value:** (GDAL >= 3.7)
- **METADATA_COPYRIGHT_YEAR=value:** (GDAL >= 3.7)
- **METADATA_DESCRIPTION=value:** (GDAL >= 3.7)
- **METADATA_KEYWORDS=value:** (GDAL >= 3.7)
- **METADATA_LINK_{N}_HREF=value:** (GDAL >= 3.7) where {N} should be substituted with a serial number (1, 2, ...)
- **METADATA_LINK_{N}_TEXT=value:** (GDAL >= 3.7) where {N} should be substituted with a serial number (1, 2, ...)
- **METADATA_LINK_{N}_TYPE=value:** (GDAL >= 3.7) where {N} should be substituted with a serial number (1, 2, ...)
- **METADATA_NAME=value:** (GDAL >= 3.7)
- **METADATA_TIME=value:** (GDAL >= 3.7)

Waypoints, routes and tracks must be written into that order to be valid against the XML Schema.

When translating from a source dataset, it may be necessary to rename the field names from the source dataset to the expected GPX attribute names, such as <name>, <desc>, etc... This can be done with a *OGR VRT* dataset, or by using the "-sql" option of the ogr2ogr utility.

7.31.9 Issues when translating to Shapefile

- When translating the *track_points* layer to a Shapefile, the field names "track_seg_id" and "track_seg_point_id" are truncated to 10 characters in the .DBF file, thus leading to duplicate names.

  To avoid this, you can define the *GPX_SHORT_NAMES* configuration option to TRUE.

  To translate the track_points layer of a GPX file to a set of shapefiles :

  ```
  ogr2ogr --config GPX_SHORT_NAMES YES out input.gpx track_points
  ```

- Shapefile does not support fields of type DateTime. It only supports fields of type Date. So by default, you will lose the hour:minute:second part of the *Time* elements of a GPX file.

  You can use the OGR SQL CAST operator to convert the *time* field to a string :

```
ogr2ogr out input.gpx -sql "SELECT ele, CAST(time AS character(32)) FROM waypoints"
```

There is a more convenient way to select all fields and ask for the conversion of the ones of a given type to strings:

```
ogr2ogr out input.gpx -fieldTypeToString DateTime
```

7.31.10 VSI Virtual File System API support

The driver supports reading and writing to files managed by VSI Virtual File System API, which include "regular" files, as well as files in the /vsizip/ (read-write) , /vsigzip/ (read-write) , /vsicurl/ (read-only) domains.

Writing to /dev/stdout or /vsistdout/ is also supported.

7.31.11 Examples

7.31.11.1 Example 1: Dumping the contents of a GPX datafile

```
ogrinfo -ro -al input.gpx
```

7.31.11.2 Example 2: GPX to GPX translation

```
ogr2ogr -f GPX output.gpx input.gpx waypoints routes tracks
```

Note: in the case of GPX to GPX translation, you need to specify the layer names, in order to discard the route_points and track_points layers.

7.31.11.3 Example 3: Use of the <extensions> tag for output

```
  ogr2ogr -f GPX  -dsco GPX_USE_EXTENSIONS=YES output.gpx input

which will give an output like the following one :
```

```
<?xml version="1.0"?>
<gpx version="1.1" creator="GDAL 1.5dev"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ogr="http://osgeo.org/gdal"
xmlns="http://www.topografix.com/GPX/1/1"
xsi:schemaLocation="http://www.topografix.com/GPX/1/1 http://www.topografix.com/GPX/1/1/
↪gpx.xsd">
<wpt lat="1" lon="2">
<extensions>
    <ogr:Primary_ID>PID5</ogr:Primary_ID>
    <ogr:Secondary_ID>SID5</ogr:Secondary_ID>
</extensions>
</wpt>
<wpt lat="3" lon="4">
<extensions>
    <ogr:Primary_ID>PID4</ogr:Primary_ID>
    <ogr:Secondary_ID>SID4</ogr:Secondary_ID>
</extensions>
</wpt>
</gpx>
```

7.31.11.4 Example 4: Use of ogr2ogr -sql option to remap field names to the ones allowed by the GPX schema

```
ogr2ogr -f GPX output.gpx input.shp -sql "SELECT field1 AS name, field2 AS desc FROM␣
↪input"
```

7.31.12 FAQ

How to solve "ERROR 6: Cannot create GPX layer XXXXXX with unknown geometry type" ?

This error happens when the layer to create does not expose a precise geometry type, but just a generic wkbUnknown type. This is for example the case when using ogr2ogr with a SQL request to a PostgreSQL datasource. You must then explicitly specify -nlt POINT (or LINESTRING or MULTILINESTRING).

7.31.13 See Also

- Home page for GPX format
- GPX 1.1 format documentation

7.32 GRASS Vector Format

Driver short name

GRASS

Build dependencies

gdal-grass-driver

The GDAL-GRASS driver can read GRASS GIS vector maps. The driver is maintained at the GDAL-GRASS GIS driver repository.

For the driver documentation, please visit the GDAL-GRASS GIS vector driver page.

7.32.1 See Also

- GRASS GIS home page

7.33 GTFS - General Transit Feed Specification

Added in version 3.7.

Driver short name

GTFS

Driver built-in by default

This driver is built-in by default

This driver can read a .zip archive containing text files following the GTFS specification.

The driver presents layers with their original structure, and:

- enhances the `stops` layer with Point geometries
- enhances the `trips` layer with LineString geometries reconstructed by joining the `trips` layer with `stops` and `stop_times` layers
- expose a `shapes_geom` layer, if the optional `shapes` is present by constructing LineString geometries.

7.33.1 Open syntax

The connection name can be:

- a .zip filename containing GTFS .txt files
- a directory name prefixed with `GTFS:` (potentially a `/vsizip/path/to/the.zip` filename prefixed with `GTFS:`)

Alternatively, starting with GDAL 3.10, specifying the `-if GTFS` option to command line utilities accepting it, or `GTFS` as the only value of the `papszAllowedDrivers` of *`GDALOpenEx()`*, also forces the driver to recognize the passed filename.

7.33.2 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.33.3 Examples

List the contents of a GTFS local zip file:

Bash

gdal CLI

```
ogrinfo GTFS_Realtime.zip
```

```
gdal vector info GTFS_Realtime.zip
```

List the contents of a GTFS local directory:

Bash

gdal CLI

```
# specifying the driver
ogrinfo GTFS:"/data/GTFS_Realtime"
# specifying the input format
ogrinfo /data/GTFS_Realtime -if GTFS
```

```
# specifying the driver
gdal vector info GTFS:"/data/GTFS_Realtime"
# specifying the input format
gdal vector info /data/GTFS_Realtime --input-format GTFS
```

List the contents of a GTFS file at a URL:

Bash

gdal CLI

```
ogrinfo /vsicurl/https://www.transportforireland.ie/transitData/Data/GTFS_Realtime.zip
```

```
gdal vector info /vsicurl/https://www.transportforireland.ie/transitData/Data/GTFS_
↪Realtime.zip
```

Extract the stops from the GTFS file and save into a FlatGeobuf file:

Bash

gdal CLI

```
ogr2ogr stops.fgb GTFS_Realtime.zip stops
```

```
gdal vector convert GTFS_Realtime.zip stops.fgb --input-layer stops
```

7.33.4 Links

- GTFS Wikipedia page
- GTFS.org

7.34 SAP HANA

Driver short name

HANA

Build dependencies

odbc-cpp-wrapper

This driver implements read and write access for spatial data stored in an SAP HANA database.

7.34.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

7.34.2 Connecting to a database

To connect to an SAP HANA database, use a connection string specifying the database name, with additional parameters as necessary. The HANA: prefix is used to mark the name as a HANA connection string.

```
HANA:"DRIVER=HDBODBC;DATABASE=HAN;HOST=localhost;PORT=30015;USER=mylogin;
↪PASSWORD=mypassword;SCHEMA=MYSCHEMA"
```

In this syntax each parameter setting is in the form keyword = value. Spaces around the equal sign are optional. To write an empty value, or a value containing spaces, surround it with single quotes, e.g., keyword = 'a value'. Single quotes and backslashes within the value must be escaped with a backslash, i.e., ' and \.

7.34.3 SQL statements

The HANA driver passes SQL statements directly to HANA by default, rather than evaluating them internally when using the ExecuteSQL() call on the OGRDataSource, or the -sql command option to ogr2ogr. Attribute query expressions are also passed directly through to HANA. It's also possible to request the OGR HANA driver to handle SQL commands with the *OGR SQL* engine, by passing **"OGRSQL"** string to the ExecuteSQL() method, as the name of the SQL dialect.

The HANA driver in OGR supports the OGRDataSource::StartTransaction(), OGRDataSource::CommitTransaction() and OGRDataSource::RollbackTransaction() calls in the normal SQL sense.

7.34.4 Creation Issues

The HANA driver does not support creation of new schemas, but it does allow creation of new layers (tables) within an existing schema.

7.34.4.1 Dataset Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are supported:

- **DSN=value:** Data source name.
- **DRIVER=value:** Name or a path to a driver. For example, *`DRIVER=HDBODBC`* (Windows) or *`DRIVER=/usr/sap/hdbclient/libodbcHDB.so`* (Linux/MacOS).
- **HOST=value:** Server host name.
- **PORT=value:** Port number.
- **USER=value:** User name.
- **PASSWORD=value:** User password.
- **DATABASE=value:** Database name.
- **USER_STORE_KEY=value:** (GDAL >= 3.8) Key defined in the SAP HANA user store (hdbuserstore).
- **SCHEMA=value:** Specifies schema used for tables listed in *`TABLES`* option.
- **TABLES=value:** Restricted set of tables to list (comma separated).

- **ENCRYPT=[YES/NO]:** Defaults to `NO`. Enables or disables TLS/SSL encryption.
- **SSL_CRYPTO_PROVIDER=[commoncrypto/sapcrypto/openssl]:** Cryptographic library provider used for SSL communication.
- **SSL_KEY_STORE=value:** Path to the keystore file that contains the server's private key.
- **SSL_TRUST_STORE=value:** Path to trust store file that contains the server's public certificate(s) (OpenSSL only).
- **SSL_VALIDATE_CERTIFICATE=[YES/NO]:** Defaults to `YES`. If set to true, the server's certificate is validated.
- **SSL_HOST_NAME_IN_CERTIFICATE=value:** Host name used to verify server's identity validated.
- **CONNECTION_TIMEOUT=<milliseconds>:** Defaults to `0`. Connection timeout measured in milliseconds. The default value is 0 (disabled).
- **PACKET_SIZE=<bytes>:** Defaults to `1` MB. Sets the maximum size of a request packet sent from the client to the server, in bytes. The minimum is 1 MB.
- **SPLIT_BATCH_COMMANDS=[YES/NO]:** Defaults to `YES`. Allows split and parallel execution of batch commands on partitioned tables.
- **DETECT_GEOMETRY_TYPE=[YES/NO]:** Defaults to `YES`. Specifies whether to detect the type of geometry columns. Note, the detection may take a significant amount of time for large tables.

7.34.4.2 Dataset Creation Options

None

7.34.4.3 Layer Creation Options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALDatasetCreateLayer()`* (C) or *`Dataset.CreateLayer`* (Python). The following layer creation options are supported:

- **OVERWRITE=[YES/NO]:** Defaults to `NO`. This may be "YES" to force an existing layer of the desired name to be destroyed before creating the requested layer.
- **LAUNDER=[YES/NO]:** Defaults to `YES`. This may be "YES" to force new fields created on this layer to have their field names "laundered" into a form more compatible with HANA. This converts to upper case and converts some special characters like "-" and "#" to "_". If "NO" exact names are preserved. If enabled the table (layer) name will also be laundered.
- **PRECISION=[YES/NO]:** Defaults to `YES`. This may be "YES" to force new fields created on this layer to try and represent the width and precision information, if available using DECIMAL(width,precision) or CHAR(width) types. If "NO" then the types REAL, INTEGER and VARCHAR will be used instead.
- **DEFAULT_STRING_SIZE=value:** Defaults to `256`. Specifies default string column size.
- **GEOMETRY_NAME=value:** Defaults to `GEOMETRY`. Specifies the name of the geometry column in new table.
- **GEOMETRY_NULLABLE=[YES/NO]:** Defaults to `YES`. Specifies whether the values of the geometry column can be NULL or not.
- **SRID=value:** Specifies the SRID of the layer.
- **FID=value:** Defaults to `OGR_FID`. Specifies the name of the FID column to create.
- **FID64=[YES/NO]:** Defaults to `NO`. Specifies whether to create the FID column with BIGINT type to handle 64bit wide ids.

- **COLUMN_TYPES=value:** Specifies a comma-separated list of strings in the format field_name=hana_field_type that define column types.
- **BATCH_SIZE=<bytes>:** Defaults to `4194304` (4MB). Specifies the number of bytes to be written per one batch.

7.34.4.4 Multitenant Database Containers

In order to connect to a tenant database, you need to specify a port number assigned exactly to a desired instance. This port number can be determined by executing the following query from the tenant database.

```
SELECT SQL_PORT FROM SYS.M_SERVICES WHERE ((SERVICE_NAME='indexserver' and␣
↪COORDINATOR_TYPE= 'MASTER') or (SERVICE_NAME='xsengine'))
```

For more details, see **Section 2.9 Connections for Multitenant Database Containers** in SAP HANA Multitenant Database Containers.

7.34.5 Examples

7.34.5.1 Example 1: List HANA layers on a specified host using ogrinfo

```
ogrinfo -ro HANA:"DRIVER=HDBODBC;DATABASE=HAN;HOST=localhost;PORT=30015;USER=mylogin;
↪PASSWORD=mypassword;SCHEMA=MYSCHEMA"
```

or

```
ogrinfo -ro HANA:"DSN=MYHANADB;USER=mylogin;PASSWORD=mypassword;SCHEMA=MYSCHEMA"
```

or

```
ogrinfo -ro HANA:"DRIVER=HDBODBC;USER_STORE_KEY=mykey;SCHEMA=MYSCHEMA"
```

7.34.5.2 Example 2: Print summary information about a the planet_osm_line layer using ogrinfo

```
$ ogrinfo -ro HANA:"DRIVER=HDBODBC;DATABASE=HAN;HOST=localhost;PORT=30015;USER=mylogin;
↪PASSWORD=mypassword;SCHEMA=MYSCHEMA" -so "planet_osm_line"

Layer name: planet_osm_line
Geometry: Line String
Feature Count: 81013
Extent: (732496.086304, 6950959.464783) - (1018694.144531, 7204272.976379)
Layer SRS WKT:
PROJCS["WGS 84 / Pseudo-Mercator",
    GEOGCS["WGS 84",
        DATUM["WGS_1984",
            SPHEROID["WGS 84",6378137,298.257223563, AHORITY["EPSG","7030"]],
            AUTHORITY["EPSG","6326"]],
            PRIMEM["Greenwich",0, AUTHORITY["EPSG","8901"]],
            UNIT["degree",0.0174532925199433, AUTHORITY["EPSG","9122"]],
            AUTHORITY["EPSG","4326"]],
        PROJECTION["Mercator_1SP"],
        PARAMETER["central_meridian",0],
        PARAMETER["scale_factor",1],
        PARAMETER["false_easting",0],
```

(continues on next page)

(continued from previous page)

```
        PARAMETER["false_northing",0],
        UNIT["metre",1,AUTHORITY["EPSG","9001"]],
        AXIS["X",EAST],
        AXIS["Y",NORTH],
        AUTHORITY["EPSG","3857"]]
Geometry Column = way
osm_id: Integer64 (0.0)
access: String (4000.0)
addr:housename: String (4000.0)
addr:housenumber: String (4000.0)
addr:interpolation: String (4000.0)
admin_level: String (4000.0)
aerialway: String (4000.0)
aeroway: String (4000.0)
```

7.34.5.3 Example 3: Export data from the points table to a shapefile called points_output.shp

```
ogr2ogr -f "ESRI Shapefile" "D:\\points_output.shp" HANA:"DRIVER=HDBODBC;DATABASE=HAN;
↪HOST=localhost;PORT=30015;USER=mylogin;PASSWORD=mypassword;SCHEMA=GIS;TABLES=points"
```

7.34.5.4 Example 4: Create and populate a table with data taken from a shapefile

```
ogr2ogr -f HANA HANA:"DRIVER=HDBODBC;DATABASE=HAN;HOST=localhost;PORT=30015;USER=mylogin;
↪PASSWORD=mypassword;SCHEMA=MYSCHEMA" myshapefile.shp
```

7.34.6 For developers

To compile the SAP HANA driver, odbc-cpp-wrapper library needs to be linked or installed. For more details, see comments in nmake.opt or configure.ac files to build the driver for Windows or Linux/MacOS correspondingly.

7.34.7 See Also

- SAP HANA Home Page
- SAP HANA Spatial Reference
- SAP HANA ODBC Connection Properties

7.35 IDB

Driver short name

IDB

Build dependencies

Informix DataBlade

This driver implements support for access to spatial tables in IBM Informix extended with the DataBlade spatial module.

When opening a database, its name should be specified in the form

```
"IDB:dbname={dbname} server={host} user={login} pass={pass} table={layertable}".
```

The IDB: prefix is used to mark the name as a IDB connection string.

If the *geometry_columns* table exists, then all listed tables and named views will be treated as OGR layers. Otherwise all regular user tables will be treated as layers.

Regular (non-spatial) tables can be accessed, and will return features with attributes, but not geometry. If the table has a "st_*" field, it will be treated as a spatial table. The type of the field is inspected to determine how to read it.

Driver supports automatic FID detection.

7.35.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.35.2 Environment variables

- `INFORMIXDIR` It should be set to Informix client SDK install dir
- `INFORMIXSERVER`: Default Informix server name
- `DB_LOCALE`: Locale of Informix database
- `CLIENT_LOCALE`: Client locale

For more information about Informix variables read documentation of Informix Client SDK

7.35.3 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). The following configuration options are supported:

- **IDB_OGR_FID=value:** Defaults to `ogc_fid`. Set name of primary key.

7.35.4 Examples

7.35.4.1 Example 1

List Informix DataBlade layers on a different hos

```
ogrinfo -ro IDB:"server=demo_on user=informix dbname=frames"
```

7.36 Idrisi Vector (.VCT)

Driver short name

IDRISI

Driver built-in by default

This driver is built-in by default

This driver reads Idrisi vector files with .vct extension. The driver recognized point, lines and polygons geometries.

For geographical referencing identification, the .vdc file contains information that points to a file that holds the geographic reference details. Those files uses extension REF and resides in the same folder as the RST image or more likely in the Idrisi installation folders.

Therefore the presence or absence of the Idrisi software in the running operation system will determine the way that this driver will work. By setting the environment variable IDRISIDIR pointing to the Idrisi main installation folder will enable GDAL to find more detailed information about geographical reference and projection in the REF files.

Note that the driver recognizes the name convention used in Idrisi for UTM and State Plane geographic reference so it doesn't need to access the REF files. That is the case for RDC file that specify "utm-30n" or "spc87ma1" in the "ref. system" field. Note that exporting to RST in any other geographical reference system will generate a suggested REF content in the comment section of the RDC file.

The driver can retrieve attributes from .ADC / .AVL ASCII files.

7.36.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.37 "INTERLIS 1" and "INTERLIS 2" drivers

Driver short name

INTERLIS 1

Driver short name

INTERLIS 2

Build dependencies

Xerces

OGR has support for INTERLIS reading.
INTERLIS is a standard which has been especially composed in order to fulfill the requirements of modeling and the integration of geodata into contemporary and future geographic information systems. With the usage of unified, documented geodata and the flexible exchange possibilities the following advantage may occur:

- the standardized documentation
- the compatible data exchange
- the comprehensive integration of geodata e.g. from different data owners.
- the quality proofing
- the long term data storage
- the contract-proof security and the availability of the software

OGR supports INTERLIS 1 and INTERLIS 2 (2.2 and newer) with the following limitations:

- Curves in Interlis 1 area polygons are converted to line segments
- Interlis 1 Surface geometries with non-numeric IDENT field are not included in the attribute layer
- Embedded INTERLIS 2 structures and line attributes are not supported
- Incremental transfer is not supported
- Transfer id (TID) is used as feature id

7.37.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.37.2 Dataset open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to `GDALOpenEx()` (C) or `gdal.OpenEx` (Python). The following open options are available:

- **MODEL=value:** Path to IlisMeta model file.

7.37.3 Model support

Data is read from transfer files which have different formats in INTERLIS 1 (.itf) and INTERLIS 2 (.xtf). Models are passed in IlisMeta format with the open option `MODEL` or by using "a_filename.xtf,models.imd" as a connection string.

IlisMeta files can be be generated with the ili2c compiler. Command line example:

```
java -jar ili2c.jar --ilidirs '%ILI_DIR;http://models.interlis.ch/;%JAR_DIR' -oIMD --out␣
↪models.imd model1.ili [model2.ili ...]
```

or for newer models:

```
java -jar ili2c.jar --ilidirs '%ILI_DIR;http://models.interlis.ch/;%JAR_DIR' -oIMD16 --
↪out models.imd model1.ili [model2.ili ...]
```

Some possible transformations using *ogr2ogr*.

- Interlis 1 -> Shape:

  ```
  gdal vector convert --oo MODEL=Beispiel.imd -f "ESRI Shapefile" ili-bsp.itf shpdir
  ```

- Interlis 2 -> GeoPackage:

  ```
  gdal vector convert --oo MODEL=KGKCGC_FPDS2_V1_1.imd fpds2_v1_1.xtf fpds2_v1_1.gpkg
  ```

  or without model:

  ```
  gdal vector convert -f "ESRI Shapefile" RoadsExdm2ien.xml shpdir
  ```

  Example with curves and multiple geometries:

  ```
  gdal vector sql --oo MODEL=OeREBKRM09vs.imd --config OGR_STROKE_CURVE=TRUE --sql
  ↪'SELECT Rechtsstatus,publiziertAb,MetadatenGeobasisdaten,Eigentumsbeschraenkung,
  ↪ZustaendigeStelle,Flaeche FROM "OeREBKRM09trsfr.Transferstruktur.Geometrie"' -f
  ↪"ESRI Shapefile" ch.bazl.sicherheitszonenplan.oereb_20131118.xtf shpdir
  ```

- Importing multiple Interlis 1 files into PostGIS:

  ```
  gdal vector convert --oo MODEL=av.imd --overwrite-layer av_fixpunkte_ohne_
  ↪LFPNachfuehrung.itf PG:dbname=warmerda
  gdal vector convert --oo MODEL=av.imd --append av_fixpunkte_mit_LFPNachfuehrung.itf␣
  ↪PG:dbname=warmerda
  ```

7.37.3.1 Arc interpolation

Converting INTERLIS arc geometries to line segments can be forced by setting the configuration variable *OGR_STROKE_CURVE* to TRUE.

The approximation of arcs as linestrings is done by splitting the arcs into subarcs of no more than a threshold angle. This angle is the *OGR_ARC_STEPSIZE*. This defaults to one degree, but may be overridden by setting the configuration variable *OGR_ARC_STEPSIZE*.

7.37.4 Other Notes

- Development of the OGR INTERLIS driver was supported by Swiss Federal Administration, Canton Solothurn and Canton Thurgovia.

7.38 JML: OpenJUMP JML format

Driver short name

JML

Build dependencies

(read support needs libexpat)

OGR has support for reading and writing .JML files used by the OpenJUMP software. Read support is only available if GDAL is built with *expat* library support

.jml is a variant of GML format. There is no formal definition of the format. It supports a single layer per file, mixed geometry types, and for each feature, a geometry and several attributes of type integer, double, string, date or object. That object data type, used for example to store 64 bit integers, but potentially arbitrary serialized Java objects, is converted as string when reading. Contrary to GML, the definition of fields is embedded in the .jml file, at its beginning.

Support for reading and writing spatial reference systems requires GDAL 2.3 or later.

7.38.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.38.2 Encoding issues

Expat library supports reading the following built-in encodings :

- US-ASCII
- UTF-8
- UTF-16
- ISO-8859-1

- Windows-1252

The content returned by OGR will be encoded in UTF-8, after the conversion from the encoding mentioned in the file header is. But files produced by OpenJUMP are always UTF-8 encoded.

When writing a JML file, the driver expects UTF-8 content to be passed in.

7.38.3 Styling

OpenJUMP uses an optional string attribute called "R_G_B" to determine the color of objects. The field value is "RRGGBB" where RR, GG, BB are respectively the value of the red, green and blue components expressed as hexadecimal values from 00 to FF. When reading a .jml file, OGR will translate the R_G_B attribute to the Feature Style encoding, unless a OGR_STYLE attribute is present. When writing a .jml file, OGR will extract from the Feature Style string the color of the PEN tool or the forecolor of the BRUSH tool to write the R_G_B attribute, unless the R_G_B attribute is defined in the provided feature. The addition of the R_G_B attribute can be disabled by setting the CREATE_R_G_B_FIELD layer creation option to NO.

7.38.4 Creation Issues

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALDatasetCreateLayer()`* (C) or *`Dataset.CreateLayer`* (Python). The JML writer supports the following *layer* creation options:

- **CREATE_R_G_B_FIELD=[YES/NO]:** Defaults to `YES`. whether the create a R_G_B field that will contain the color of the PEN tool or the forecolor of the BRUSH tool of the OGR Feature Style string.
- **CREATE_OGR_STYLE_FIELD=[YES/NO]:** Defaults to `NO`. whether the create a OGR_STYLE field that will contain the Feature Style string.

7.38.5 See Also

- *Feature Style Specification*

7.38.6 Credits

The author wishes to thank Jukka Rahkonen for funding the development of this driver.

7.39 JSONFG -- OGC Features and Geometries JSON

Added in version 3.8.

Driver short name

JSONFG

Driver built-in by default

This driver built-in by default

This driver implements read/write support for access to features encoded in OGC Features and Geometries JSON - Part 1: Core format.

JSON-FG is a set of extensions over the *GeoJSON* format.

The following extensions are supported:

- capturing the feature type in a `featureType` element to affect features to separate
- capturing a coordinate reference system (not necessarily WGS 84) in a `coordRefSys` element, that is the one used by geometries written in the `place` element
- `time` element at Feature level
- minimum support for Polyhedron geometries (with a single outer shell) and Prism with Point, LineString or Polygon base.

Warning

This driver implements a draft version of the standard.

7.39.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.39.2 Connection string

The JSON-FG driver accepts three types of sources of data:

- Uniform Resource Locator (URL) - a Web address to perform HTTP request
- Plain text file with JSON-FG data, typically with a .json extension
- Text passed directly and encoded in JSON-FG

The URL/filename/text might be prefixed with `JSONFG:` to avoid any ambiguity with other drivers. Alternatively, starting with GDAL 3.10, specifying the `-if JSONFG` option to command line utilities accepting it, or `JSONFG` as the only value of the `papszAllowedDrivers` of *GDALOpenEx()*, also forces the driver to recognize the passed URL/filename/text.

7.39.3 CRS support

Starting with GDAL 3.12, if writing a CRS which has no known identifier, the CRS definition will be written as a PROJJSON object like:

```
"coordRefSys": {
    "type": "PROJJSON",
    "value": {
        "$schema": "https://proj.org/schemas/v0.7/projjson.schema.json",
```

(continues on next page)

(continued from previous page)

```
        "type": "CRS type",
        "name": "CRS name",
        "...[snip]...": {}
    }
}
```

7.39.4 Time support

On reading:

- Feature.time.date elements are mapped to a OGR `time` field of type Date. If there is a property also called `time`, the OGR field for Feature.time is called `jsonfg_time`
- Feature.time.timestamp elements are mapped to a OGR `time` field of type DateTime
- Feature.time.interval elements are mapped to a OGR `time_start` and `time_end` fields of type Date or DateTime. If there are properties with those names, the OGR fields for Feature.time are prefixed with `jsonfg_` Unbounded intervals (`..`) are mapped to a unset OGR feature field.

On writing, if fields called `time`, `time_start`, `time_end` of type Date or DateTime are created, their content is used to set the Feature.time element with the reverse logic as the above explained reading side. The field names may also be prefixed with `jsonfg_` to distinguish them from fields `time`, `time_start`, `time_end` that must be written in Feature.properties

7.39.5 Circular-arc geometry support

Added in version 3.12.

Reading and writing of CircularString, CompoundCurve, CurvePolygon, MultiCurve and MultiSurface geometries is supported.

7.39.6 Geometry measure support

Added in version 3.12.

Geometry with measure values (M component) can be read and written. The unit and description of the measure, if included in the file, is reported in the `MEASURES` layer metadata domain. On the writing side, the *`MEASURE_UNIT`* and *`MEASURE_DESCRIPTION`* layer creation option can be set.

7.39.7 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The following configuration options are available:

- **OGR_JSONFG_MAX_OBJ_SIZE=<MBytes>:** Defaults to `200`. size in MBytes of the maximum accepted single feature, or 0 to allow for a unlimited size (GDAL >= 3.5.2).

7.39.8 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are supported:

- **GEOMETRY_ELEMENT=[AUTO/PLACE/GEOMETRY]:** Defaults to `AUTO`. Determines which JSON element to use to create geometry from. By default when the `place` element is set, geometries are created from its content. And when it is absent or null, the content of the `geometry` element is used. Setting this option to

PLACE restricts the reading to the place elements. Setting this option to GEOMETRY restricts the reading to the geometry elements.

7.39.9 Dataset creation options

Dataset creation options can be specified in command-line tools using the syntax -dsco <NAME>=<VALUE> or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following dataset creation options are supported:

- **SINGLE_LAYER=[YES/NO]:** Defaults to NO. Whether a single layer will be written. If set to YES, the coordRefSys and featureType elements can be written at the FeatureCollection level instead inside each Feature.

7.39.10 Layer creation options

Layer creation options can be specified in command-line tools using the syntax -lco <NAME>=<VALUE> or by providing the appropriate arguments to *GDALDatasetCreateLayer()* (C) or *Dataset.CreateLayer* (Python). The following layer creation options are supported:

- **COORDINATE_PRECISION_GEOMETRY=<integer>:** Defaults to 7. Maximum number of figures after decimal separator to write in coordinates of geometry element.
- **COORDINATE_PRECISION_PLACE=<integer>:** Maximum number of figures after decimal separator to write in coordinates of place element.
- **SIGNIFICANT_FIGURES=<integer>:** Defaults to 17. Maximum number of significant figures when writing floating-point numbers. If explicitly specified, and *COORDINATE_PRECISION_GEOMETRY* or *COORDINATE_PRECISION_PLACE* is not, this will also apply to coordinates.
- **WRITE_GEOMETRY=[YES/NO]:** Defaults to YES. By default, when a non-WGS 84 CRS is assigned to the layer, the geometry is written in the place element, and a reprojected version of it to WGS 84 is written in the geometry element. This option can be set to NO to avoid writing the geometry element when place is written. Setting this option to NO has no effect if the layer CRS is WGS 84, in which case geometries must be written in geometry.
- **ID_FIELD=value:** Name of the source field that must be written as the 'id' member of Feature objects.
- **ID_TYPE=[AUTO/String/Integer]:** Type of the 'id' member of Feature objects.
- **ID_GENERATE=[YES/NO]:** Auto-generate feature ids
- **MEASURE_UNIT=<string>:** (GDAL >= 3.12) Unit of measures (M) values
- **MEASURE_DESCRIPTION=<string>:** (GDAL >= 3.12) Description of measures (M) values

7.39.11 VSI Virtual File System API support

The driver supports reading and writing to files managed by VSI Virtual File System API, which includes "regular" files, as well as files in the /vsizip/ (read-write), /vsigzip/ (read-write), /vsicurl/ (read-only) domains.

Writing to /dev/stdout or /vsistdout/ is also supported.

7.39.12 Geometry coordinate precision

Added in version GDAL: 3.9

The GeoJSON driver supports reading and writing the geometry coordinate precision, using the *OGRGeomCoordinatePrecision* settings of the *OGRGeomFieldDefn* Those settings are used to round the coordinates of the geometry of the features to an appropriate decimal precision.

Note

The *COORDINATE_PRECISION_GEOMETRY* or *COORDINATE_PRECISION_PLACE* layer creation option has precedence over the values set on the *OGRGeomFieldDefn*.

Implementation details: the coordinate precision is stored as `xy_coordinate_resolution_place` and `z_coordinate_resolution_place` members at the FeatureCollection level, for the geometries written in the `place` element. Their numeric value is expressed in the units of the SRS.

For the `geometry` standard GeoJSON element, the coordinate precision is stored as `xy_coordinate_resolution` and `z_coordinate_resolution` members, and their numeric value is expressed in the units of the OGC:CRS84 SRS (hence decimal degrees for `xy_coordinate_resolution`)

Example:

```
{
    "type": "FeatureCollection",
    "xy_coordinate_resolution_place": 1.0,
    "z_coordinate_resolution_place": 1.0,
    "xy_coordinate_resolution": 8.9e-6,
    "z_coordinate_resolution": 1e-1,
    "features": []
}
```

7.39.13 See Also

- OGC Features and Geometries JSON - Part 1: Core
- *GeoJSON driver*

7.40 KML - Keyhole Markup Language

Driver short name

KML

Build dependencies

(read support needs libexpat)

Keyhole Markup Language (KML) is an XML-based language for managing the display of 3D geospatial data. KML has been accepted as an OGC standard, and is supported in one way or another on the major GeoBrowsers. Note that KML by specification uses only a single projection, EPSG:4326. All OGR KML output will be presented in EPSG:4326. As such OGR will create layers in the correct coordinate system and transform any geometries.

At this time, only vector layers are handled by the KML driver. *(there are additional scripts supplied with the GDAL project that can build other kinds of output)*

7.40.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.40.1.1 KML Reading

KML reading is only available if GDAL/OGR is built with the Expat XML Parser, otherwise only KML writing will be supported.

Supported geometry types are `Point`, `Linestring`, `Polygon`, `MultiPoint`, `MultiLineString`, `MultiPolygon` and `MultiGeometry`. There are limitations, for example: the nested nature of folders in a source KML file is lost; folder `<description>` tags will not carry through to output. Folders containing multiple geometry types, like POINT and POLYGON, are supported.

7.40.1.2 KML Writing

Since not all features of KML are able to be represented in the Simple Features geometry model, you will not be able to generate many KML-specific attributes from within GDAL/OGR. Please try a few test files to get a sense of what is possible.

When outputting KML, the OGR KML driver will translate each OGR Layer into a KML Folder (you may encounter unexpected behavior if you try to mix the geometry types of elements in a layer, e.g. `LINESTRING` and `POINT` data).

The KML Driver will rename some layers, or source KML folder names, into new names it considers valid, for example '`Layer #0`', the default name of the first unnamed Layer, becomes `'Layer__0'`.

KML is mix of formatting and feature data. The <description> tag of a Placemark will be displayed in most geobrowsers as an HTML-filled balloon. When writing KML, Layer element attributes are added as simple schema fields. This best preserves feature type information.

Limited support is available for fills, line color and other styling attributes. Please try a few sample files to get a better sense of actual behavior.

7.40.1.3 Encoding issues

Expat library supports reading the following built-in encodings :

- US-ASCII
- UTF-8
- UTF-16
- ISO-8859-1
- Windows-1252

The content returned by OGR will be encoded in UTF-8, after the conversion from the encoding mentioned in the file header is.

If your KML file is not encoded in one of the previous encodings, it will not be parsed by the KML driver. You may convert it into one of the supported encoding with the *iconv* utility for example and change accordingly the *encoding* parameter value in the XML header.

When writing a KML file, the driver expects UTF-8 content to be passed in.

7.40.1.4 Creation Options

Dataset creation options can be specified in command-line tools using the syntax `-dsco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following dataset creation options are supported:

- **NameField=value:** Defaults to `Name`. Allows you to specify the field to use for the KML <name> element.
- **DescriptionField=value:** Defaults to `Description`. Allows you to specify the field to use for the KML <description> element.
- **AltitudeMode=value:** Allows you to specify the AltitudeMode to use for KML geometries. This will only affect 3D geometries and must be one of the valid KML options. See the relevant KML reference material for further information.

```
ogr2ogr -f KML output.kml input.shp -dsco AltitudeMode=absolute
```

- **DOCUMENT_ID=value:** Defaults to `root_doc`. Specifies the id of the root <Document> node.

7.40.2 VSI Virtual File System API support

The driver supports reading and writing to files managed by VSI Virtual File System API, which include "regular" files, as well as files in the /vsizip/ (read-write) , /vsigzip/ (read-write) , /vsicurl/ (read-only) domains.

Writing to /dev/stdout or /vsistdout/ is also supported.

7.40.3 Examples

7.40.3.1 Example 1: Writing the results of a PostGIS query to KML

```
ogr2ogr -f KML output.kml PG:'host=myserver dbname=warmerda' -sql "SELECT pop_1994 from␣
↪canada where province_name = 'Alberta'"
```

7.40.3.2 Example 2: Dumping the contents of a .kml file as OGR sees it

```
ogrinfo -ro somedisplay.kml
```

7.40.4 Caveats

Google Earth seems to have some limits regarding the number of coordinates in complex geometries like polygons. If the problem appears, then problematic geometries are displayed completely or partially covered by vertical stripes. Unfortunately, there are no exact number given in the KML specification about this limitation, so the KML driver will not warn about potential problems. One of possible and tested solutions is to simplify a line or a polygon to remove some coordinates. Here is the whole discussion about this issue on the Google KML Developer Forum, in the polygon displays with vertical stripes thread.

Also note this driver is **not** KMZ-aware. See Also --------

- KML Specification
- KML Tutorial
- *LIBKML driver* An alternative GDAL KML driver

7.41 LIBKML Driver (.kml .kmz)

Driver short name

LIBKML

Build dependencies

libkml

The LIBKML driver is a client of Libkml , a reference implementation of KML reading and writing, in the form of a cross platform C++ library. You must build and install Libkml in order to use this OGR driver. Note: you need to build libkml 1.3 or master.

Note that if you build and include this LIBKML driver, it will become the default reader of KML for ogr, overriding the previous *KML driver*. You can still specify either KML or LIBKML as the output driver via the command line

Libkml from Google provides reading services for any valid KML file. However, please be advised that some KML facilities do not map into the Simple Features specification OGR uses as its internal structure. Therefore, a best effort will be made by the driver to understand the content of a KML file read by libkml into ogr, but your mileage may vary. Please try a few KML files as samples to get a sense of what is understood. In particular, nesting of feature sets more than one deep will be flattened to support ogr's internal format.

7.41.1 Driver capabilities

Supports Create()

This driver supports the *`GDALDriver::Create()`* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.41.2 Datasource

You may specify a datasource as a kml file `somefile.kml` , a directory `somedir/` , or a kmz file `somefile.kmz` .

By default on directory and kmz datasources, an index file of all the layers will be read from or written to doc.kml. It contains a <NetworkLink> to each layer file in the datasource. This feature can be turned off by setting the configuration option *LIBKML_USE_DOC.KML* to "NO":

- **LIBKML_USE_DOC.KML=[YES/NO]:** Defaults to `YES`. Use a `doc.kml` index file.

7.41.2.1 StyleTable

Datasource style tables are written to the <Document> in a .kml, style/style.kml in a kmz file, or style.kml in a directory, as one or more <Style> elements. Not all of *Feature Style Specification* can translate into KML.

7.41.2.2 Dataset creation options

Dataset creation options can be specified in command-line tools using the syntax `-dsco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following dataset creation options can be used to generate a <atom:Author> element at the top Document level.

- **AUTHOR_NAME=value:** Specifies the value of the `<atom:name>` element.
- **AUTHOR_URI=value:** Specifies the value of the `<atom:uri>` element. It should start with `http://` or `https://`
- **AUTHOR_EMAIL=value:** Specifies the value of the `<atom:email>` element. It should include a `@` character.

Additional dataset creation options affecting the top Document level:

- **LINK=value:** Specifies the href of an <atom:link> element at the top Document level.
- **PHONENUMBER=value:** Specifies the value of the <phoneNumber> element at the top Document level. The value must follow the syntax of IETF RFC 3966.
- **DOCUMENT_ID=value:** Defaults to `root_doc`. Specifies the id of the root <Document> node.

Container properties

The following dataset creation options can be used to set container options :

- **NAME=value:** <name> element
- **VISIBILITY=value:** <visibility> element
- **OPEN=value:** <open> element
- **SNIPPET=value:** <snippet> element
- **DESCRIPTION=value:** <description> element

List style

The following dataset creation options can be used to control how the main folder (folder of layers) appear in the Places panel of the Earth browser, through a <ListStyle> element:

- **LISTSTYLE_TYPE=[check/radioFolder/checkOffOnly/checkHideChildren]:** Sets the <listItemType> element.
- **LISTSTYLE_ICON_HREF=value:** URL of the icon to display for the main folder. Sets the href element of the <ItemIcon> element.

Balloon style

If a style *foo* is defined, it is possible to add a <BalloonStyle> element to it, by specifying the **foo_BALLOONSTYLE_BGCOLOR** and/or **foo_BALLOONSTYLE_TEXT** elements.

NetworkLinkControl

A <NetworkLinkControl> element can be defined if at least one of the following dataset creation option is specified:

- **NLC_MINREFRESHPERIOD=value:** to set the <minRefreshPeriod> element
- **NLC_MAXSESSIONLENGTH=value:** to set the <maxSessionLength> element
- **NLC_COOKIE=value:** to set the <cookie> element
- **NLC_MESSAGE=value:** to set the <message> element
- **NLC_LINKNAME=value:** to set the <linkName> element
- **NLC_LINKDESCRIPTION=value:** to set the <linkDescription> element
- **NLC_LINKSNIPPET=value:** to set the <linkSnippet> element
- **NLC_EXPIRES=value:** to set the <expires> element

Update documents

When defining the dataset creation option **UPDATE_TARGETHREF**, a NetworkLinkControl KML file with an <Update> element will be generated. See the tutorial about update.

The CreateFeature() operation on a layer will be translated as a <Create> element.

The SetFeature() operation on a layer will be translated as a <Change> element.

The DeleteFeature() operation on a layer will be translated as a <Delete> element.

7.41.3 Layer

`OGRLayer` are mapped to kml files as a <Document> or <Folder>, and in kmz files or directories as a separate kml file.

7.41.3.1 Style

Layer style tables can not be read from or written to a kml layer that is a <Folder>, otherwise they are in the <Document> that is the layer.

7.41.3.2 Schema

Read and write of <Schema> is supported for .kml files, .kmz files, and directories.

7.41.3.3 Layer creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALDatasetCreateLayer()`* (C) or *`Dataset.CreateLayer`* (Python). The following layer creation options can be used to generate a <LookAt> element at the layer level.

- **LOOKAT_LONGITUDE=value:** (required)
- **LOOKAT_LATITUDE=value:** (required)
- **LOOKAT_RANGE=value:** (required)
- **LOOKAT_HEADING=value:**

- **LOOKAT_TILT=value:**
- **LOOKAT_ALTITUDE=value:**
- **LOOKAT_ALTITUDEMODE=value:**

Alternatively, a <Camera> element can be generated.

- **CAMERA_LONGITUDE=value:** (required)
- **CAMERA_LATITUDE=value:** (required)
- **CAMERA_ALTITUDE=value:** (required)
- **CAMERA_ALTITUDEMODE=value:** (required)
- **CAMERA_HEADING=value:**
- **CAMERA_TILT=value:**
- **CAMERA_ROLL=value:**

A <Region> element can be generated to control when objects of the layer are visible or not. If *REGION_XMIN*, *REGION_YMIN*, *REGION_XMAX* and *REGION_YMAX*, the region coordinates are determined from the spatial extent of the features being written in the layer.

- **ADD_REGION=[YES/NO]:** Defaults to `NO`.
- **REGION_XMIN=value:** defines the west coordinate of the region.
- **REGION_YMIN=value:** defines the south coordinate of the region.
- **REGION_XMAX=value:** defines the east coordinate of the region.
- **REGION_YMAX=value:** defines the north coordinate of the region.
- **REGION_MIN_LOD_PIXELS=value:** Defaults to `256`. minimum size in pixels of the region so that it is displayed.
- **REGION_MAX_LOD_PIXELS=value:** Defaults to `-1`. maximum size in pixels of the region so that it is displayed. Defaults to -1 (infinite).
- **REGION_MIN_FADE_EXTENT=value:** Defaults to `0`. distance over which the geometry fades, from fully opaque to fully transparent.
- **REGION_MAX_FADE_EXTENT=value:** Defaults to `0`. distance over which the geometry fades, from fully transparent to fully opaque.

A <ScreenOverlay> element can be added to display a logo, a legend, etc...

- **SO_HREF=value:** (required) URL of the image to display.
- **SO_NAME=value:**
- **SO_DESCRIPTION=value:**
- **SO_OVERLAY_X=value:**
- **SO_OVERLAY_Y=value:**
- **SO_OVERLAY_XUNITS=value:**
- **SO_OVERLAY_YUNITS=value:**
- **SO_SCREEN_X=value:** Defaults to `0.05`.
- **SO_SCREEN_Y=value:** Defaults to `0.05`.
- **SO_SCREEN_XUNITS=value:** Defaults to `Fraction`.

- **SO_SCREEN_YUNITS=value:** Defaults to `Fraction`.
- **SO_SIZE_X=value:**
- **SO_SIZE_Y=value:**
- **SO_SIZE_XUNITS=value:**
- **SO_SIZE_YUNITS=value:**

The following option controls whether layers are written as a Document or a Folder:

- **FOLDER=[YES/NO]:** By default, layers are written as <Document> elements. By setting this option to YES, it is also possible to write them as <Folder> elements (only in .kml files).

The following layer creation options can be used to set container options :

- **NAME=value:** <name> element. Also used to set the name of the NetworkLink pointing to the layer, starting with GDAL 3.10
- **VISIBILITY=value:** <visibility> element
- **OPEN=value:** <open> element
- **SNIPPET=value:** <snippet> element
- **DESCRIPTION=value:** <description> element

Do not confuse them with the same named dataset creation options.

The following layer creation options can be used to control how the folder of a layer appear in the Places panel of the Earth browser, through a <ListStyle> element:

- **LISTSTYLE_TYPE=[check/radioFolder/checkOffOnly/checkHideChildren]:** Sets the <listItemType> element.
- **LISTSTYLE_ICON_HREF=value:** URL of the icon to display for the layer folder. Sets the href element of the <ItemIcon> element.

7.41.4 Feature

An *`OGRFeature`* generally translates to kml as a <Placemark>, and vice-versa.

If the model field is defined, a <Model> object within the Placemark will be generated.

If the networklink field is defined, a <NetworkLink> will be generated. Other networklink fields are optional.

If the photooverlay field is defined, a <PhotoOverlay> will be generated (provided that the camera_longitude, camera_latitude, camera_altitude, camera_altitudemode, head and/or tilt and/or roll, leftfov, rightfov, bottomfov, topfov, near fields are also set. The shape field is optional.

In case the PhotoOverlay is a big image, it is highly recommended to tile it and generate overview levels, as explained in the PhotoOverlay tutorial. In which case, the URL should contain the "$[level]", "$[x]" and "$[y]" sub-strings in the photooverlay field, and the imagepyramid_tilesize, imagepyramid_maxwidth, imagepyramid_maxheight and imagepyramid_gridorigin fields should be set.

Placemark, Model, NetworkLink and PhotoOverlay objects can have an associated camera if the camera_longitude, camera_latitude, camera_altitude, camera_altitudemode, head and/or tilt and/or roll fields are defined.

KML <GroundOverlay> elements are supported for reading (unless the *`LIBKML_READ_GROUND_OVERLAY`* configuration option is set to FALSE). For such elements, there are icon and drawOrder fields.

The following configuration options affect reading of features:

- **LIBKML_READ_GROUND_OVERLAY=[TRUE/FALSE]:** Defaults to `TRUE`. If `FALSE`, skip reading GroundOverlay elements.

7.41.4.1 Style

Style Strings at the feature level are Mapped to KML as either a <Style> or <StyleUrl> in each <Placemark>.

The following configuration options affect handling of styles:

- **LIBKML_RESOLVE_STYLE=[YES/NO]:** When reading a kml feature and this option is set to yes, styleurls are looked up in the style tables and the features style string is set to the style from the table. This is to allow reading of shared styles by applications, like MapServer, that do not read style tables.
- **LIBKML_EXTERNAL_STYLE=[YES/NO]:** When reading a kml feature and this option is set to yes, a styleurl that is external to the datasource is read from disk or fetched from the server and parsed into the datasource style table. If the style kml can not be read or *LIBKML_EXTERNAL_STYLE=NO* then the styleurl is copied to the style string.
- **LIBKML_STYLEMAP_KEY=value:** When reading a kml StyleMap the default mapping is set to normal. If you wish to use the highlighted styles set this configuration option to "highlight"

When writing a kml, if there exist 2 styles of the form "astylename_normal" and "astylename_highlight" (where astylename is any string), then a StyleMap object will be creating from both styles and called "astylename".

7.41.5 Fields

OGR fields (feature attributes) are mapped to kml with <Schema>; and <SimpleData>, except for some special fields as noted below.

> **Note**
>
> It is also possible to export fields as <Data> elements using the following configuration option:
>
> **LIBKML_USE_SIMPLEFIELD=[YES/NO]:** If `NO`, export fields as <Data>.

A rich set of *configuration options* are available to define how fields in input and output, map to a KML <Placemark>. For example, if you want a field called 'Cities' to map to the <name>; tag in KML, you can set a configuration option. Note these are independent of layer creation and dataset creation options' *<name>*. Use an empty value to remove a field, e.g., *LIBKML_EXTRUDE_FIELD=*

- **LIBKML_ID_FIELD=value:** (GDAL >= 3.10) Defaults to `id`. Name of the string field that maps to the kml attribute <id>.
- **LIBKML_NAME_FIELD=value:** Defaults to `name`. Name of the string field that maps to the kml tag <name>.
- **LIBKML_DESCRIPTION_FIELD=value:** Defaults to `description`. Name of the string field that maps to the kml tag <description>.
- **LIBKML_TIMESTAMP_FIELD=value:** Defaults to `timestamp`. Name of the string/datetime/date/time field that maps to the kml tag <timestamp>.
- **LIBKML_BEGIN_FIELD=value:** Defaults to `begin`. Name of the string/datetime/date/time field that maps to the kml tag <begin>.
- **LIBKML_END_FIELD=value:** Defaults to `end`. Name of the string/datetime/date/time field that maps to the kml tag <end>.
- **LIBKML_ALTITUDEMODE_FIELD=value:** Defaults to `altitudeMode`. Name of the string field that maps to the kml tag <altitudeMode> or <gx:altitudeMode>.
- **LIBKML_TESSELLATE_FIELD=value:** Defaults to `tessellate`. Name of the integer field that maps to the kml tag <tessellate>.

- **LIBKML_EXTRUDE_FIELD=value:** Defaults to `extrude`. Name of the integer field that maps to the kml tag <extrude>.
- **LIBKML_VISIBILITY_FIELD=value:** Defaults to `visibility`. Name of the integer field that maps to the kml tag <visibility>.
- **LIBKML_ICON_FIELD=value:** Defaults to `icon`. Name of the string field that maps to the kml tag <icon>.
- **LIBKML_DRAWORDER_FIELD=value:** Defaults to `drawOrder`. Name of the integer field that maps to the kml tag <drawOrder>.
- **LIBKML_SNIPPET_FIELD=value:** Defaults to `snippet`. Name of the integer field that maps to the kml tag <snippet>.
- **LIBKML_HEADING_FIELD=value:** Defaults to `heading`. Name of the real field that maps to the kml tag <heading>. When reading, this field is present only if a Placemark has a Camera with a heading element.
- **LIBKML_TILT_FIELD=value:** Defaults to `tilt`. Name of the real field that maps to the kml tag <tilt>. When reading, this field is present only if a Placemark has a Camera with a tilt element.
- **LIBKML_ROLL_FIELD=value:** Defaults to `roll`. Name of the real field that maps to the kml tag <roll>. When reading, this field is present only if a Placemark has a Camera with a roll element.
- **LIBKML_MODEL_FIELD=value:** Defaults to `model`. Name of the string field that can be used to define the URL of a 3D <model>.
- **LIBKML_SCALE_X_FIELD=value:** Defaults to `scale_x`. Name of the real field that maps to the x element of the kml tag <scale> for a 3D model.
- **LIBKML_SCALE_Y_FIELD=value:** Defaults to `scale_y`. Name of the real field that maps to the y element of the kml tag <scale>for a 3D model.
- **LIBKML_SCALE_Z_FIELD=value:** Defaults to `scale_z`. Name of the real field that maps to the z element of the kml tag <scale>for a 3D model.
- **LIBKML_NETWORKLINK_FIELD=value:** Defaults to `networklink`. Name of the string field that maps to the href element of the kml tag <href> of a NetworkLink.
- **LIBKML_NETWORKLINK_REFRESHVISIBILITY_FIELD=value:** Defaults to `networklink_refreshvisibility`. Name of the integer field that maps to kml tag <refreshVisibility> of a NetworkLink.
- **LIBKML_NETWORKLINK_FLYTOVIEW_FIELD=value:** Defaults to `networklink_flytoview`. Name of the integer field that maps to kml tag <flyToView> of a NetworkLink.
- **LIBKML_NETWORKLINK_REFRESHMODE_FIELD=value:** Defaults to `networklink_refreshmode`. Name of the string field that maps to kml tag <refreshMode> of a NetworkLink.
- **LIBKML_NETWORKLINK_REFRESHINTERVAL_FIELD=value:** Defaults to `networklink_refreshinterval`. Name of the real field that maps to kml tag <refreshInterval> of a NetworkLink.
- **LIBKML_NETWORKLINK_VIEWREFRESHMODE_FIELD=value:** Defaults to `networklink_viewrefreshmode`. Name of the string field that maps to kml tag <viewRefreshMode> of a NetworkLink.
- **LIBKML_NETWORKLINK_VIEWREFRESHTIME_FIELD=value:** Defaults to `networklink_viewrefreshtime`. Name of the real field that maps to kml tag <viewRefreshTime> of a NetworkLink.

- **LIBKML_NETWORKLINK_VIEWBOUNDSCALE_FIELD=value:** Defaults to `networklink_viewboundscale`. Name of the real field that maps to kml tag <viewBoundScale> of a NetworkLink.
-
- **LIBKML_NETWORKLINK_HTTPQUERY_FIELD=value:** Defaults to `networklink_httpquery`. Name of the string field that maps to kml tag <httpQuery> of a NetworkLink.
- **LIBKML_CAMERA_LONGITUDE_FIELD=value:** Defaults to `camera_longitude`. Name of the real field that maps to kml tag <longitude> of a <Camera>.
-
- **LIBKML_CAMERA_ALTITUDE_FIELD=value:** Defaults to `camera_altitude`. Name of the real field that maps to kml tag <altitude> of a <Camera>.
- **LIBKML_CAMERA_ALTITUDEMODE_FIELD=value:** Defaults to `camera_altitudemode`. Name of the real field that maps to kml tag <altitudeMode> of a <Camera>.
- **LIBKML_PHOTOOVERLAY_FIELD=value:** Defaults to `photooverlay`. Name of the string field that maps to the href element of the kml tag <href> of a <PhotoOverlay>.
- **LIBKML_LEFTFOV_FIELD=value:** Defaults to `leftfov`. Name of the real field that maps to to kml tag <LeftFov> of a <PhotoOverlay>.
- **LIBKML_RIGHTFOV_FIELD=value:** Defaults to `rightfov`. Name of the real field that maps to to kml tag <RightFov> of a <PhotoOverlay>.
- **LIBKML_BOTTOMFOV_FIELD=value:** Defaults to `bottomfov`. Name of the real field that maps to to kml tag <BottomFov> of a <PhotoOverlay>.
- **LIBKML_TOPFOV_FIELD=value:** Defaults to `topfov`. Name of the real field that maps to to kml tag <TopFov> of a <PhotoOverlay>.
- **LIBKML_NEARFOV_FIELD=value:** Defaults to `near`. Name of the real field that maps to to kml tag <Near> of a <PhotoOverlay>.
- **LIBKML_PHOTOOVERLAY_SHAPE_FIELD=value:** Defaults to `shape`. Name of the string field that maps to to kml tag <shape> of a <PhotoOverlay>.
- **LIBKML_IMAGEPYRAMID_TILESIZE=value:** Defaults to `imagepyramid_tilesize`. Name of the integer field that maps to to kml tag <tileSize> of a <ImagePyramid>.
- **LIBKML_IMAGEPYRAMID_MAXWIDTH=value:** Defaults to `imagepyramid_maxwidth`. Name of the integer field that maps to to kml tag <maxWidth> of a <ImagePyramid>.
- **LIBKML_IMAGEPYRAMID_MAXHEIGHT=value:** Defaults to `imagepyramid_maxheight`. Name of the integer field that maps to to kml tag <maxHeight> of a <ImagePyramid>.
- **LIBKML_IMAGEPYRAMID_GRIDORIGIN=value:** Defaults to `imagepyramid_gridorigin`. Name of the string field that maps to to kml tag <gridOrigin> of a <ImagePyramid>.
- **OGR_STYLE=value:** string field that maps to a features style string, OGR reads this field if there is no style string set on the feature.

7.41.6 Geometry

Translation of *`OGRGeometry`* to KML Geometry is pretty straightforward with only a couple of exceptions. Point to <Point> (unless heading and/or tilt and/or roll field names are found, in which case a Camera object will be generated), LineString to <LineString>, LinearRing to <LinearRing>, and Polygon to <Polygon>. In OGR a polygon contains an array of LinearRings, the first one being the outer ring. KML has the tags <outerBoundaryIs> and <innerBoundaryIs> to differentiate between the two. OGR has several Multi types of geometry : GeometryCollection, MultiPolygon,

MultiPoint, and MultiLineString. When possible, OGR will try to map <MultiGeometry> to the more precise OGR geometry type (MultiPoint, MultiLineString or MultiPolygon), and default to GeometryCollection in case of mixed content.

The following configuration options control geometry translation:

- **LIBKML_WRAPDATELINE=[YES/NO]:** Sometimes kml geometry will span the dateline, In applications like QGIS or MapServer this will create horizontal lines all the way around the globe. Setting this to "yes" will cause the libkml driver to split the geometry at the dateline when read.

7.41.7 VSI Virtual File System API support

The driver supports reading and writing to files managed by VSI Virtual File System API, which include "regular" files, as well as files in the /vsizip/ (read-write) , /vsigzip/ (read-write) , /vsicurl/ (read-only) domains.

Writing to /dev/stdout or /vsistdout/ is also supported.

7.41.8 Examples

The following bash script will build a *csv* file and a *vrt* file, and then translate them to KML using *ogr2ogr* into a .kml file with timestamps and styling.

```
#!/bin/bash
# Copyright (c) 2010, Brian Case
#
# Permission is hereby granted, free of charge, to any person obtaining a
# copy of this software and associated documentation files (the "Software"),
# to deal in the Software without restriction, including without limitation
# the rights to use, copy, modify, merge, publish, distribute, sublicense,
# and/or sell copies of the Software, and to permit persons to whom the
# Software is furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included
# in all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
# OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
# THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
# FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
# DEALINGS IN THE SOFTWARE.


icon="http://maps.google.com/mapfiles/kml/shapes/shaded_dot.png"
rgba33="#FF9900"
rgba70="#FFFF00"
rgba150="#00FF00"
rgba300="#0000FF"
rgba500="#9900FF"
rgba800="#FF0000"

function docsv {

    IFS=','
```

(continues on next page)

(continued from previous page)

```
    while read Date Time Lat Lon Mag Dep
    do
        ts=$(echo $Date | sed 's:/:-:g')T${Time%%.*}Z
        rgba=""

        if [[ $rgba == "" ]] && [[ $Dep -lt 33 ]]
        then
            rgba=$rgba33
        fi

        if [[ $rgba == "" ]] && [[ $Dep -lt 70 ]]
        then
            rgba=$rgba70
        fi

        if [[ $rgba == "" ]] && [[ $Dep -lt 150 ]]
        then
            rgba=$rgba150
        fi

        if [[ $rgba == "" ]] && [[ $Dep -lt 300 ]]
        then
            rgba=$rgba300
        fi

        if [[ $rgba == "" ]] && [[ $Dep -lt 500 ]]
        then
            rgba=$rgba500
        fi

        if [[ $rgba == "" ]]
        then
            rgba=$rgba800
        fi



        style="\"SYMBOL(s:$Mag,id:\"\"$icon\"\",c:$rgba)\""

        echo $Date,$Time,$Lat,$Lon,$Mag,$Dep,$ts,"$style"
    done

}


wget http://neic.usgs.gov/neis/gis/qed.asc -O /dev/stdout |\
 tail -n +2 > qed.asc

echo Date,TimeUTC,Latitude,Longitude,Magnitude,Depth,timestamp,OGR_STYLE > qed.csv

docsv < qed.asc >> qed.csv
```

(continues on next page)

(continued from previous page)

```
cat > qed.vrt << EOF
<OGRVRTDataSource>
    <OGRVRTLayer name="qed">
        <SrcDataSource>qed.csv</SrcDataSource>
        <GeometryType>wkbPoint</GeometryType>
        <LayerSRS>WGS84</LayerSRS>
        <GeometryField encoding="PointFromColumns" x="Longitude" y="Latitude"/>
    </OGRVRTLayer>
</OGRVRTDataSource>

EOF

ogr2ogr -f libkml qed.kml qed.vrt
```

The following example shows how the three levels of <*name*> in LIBKML relate to their controlling options:

```
ogr2ogr -f LIBKML /dev/stdout 0.contours.csv \
  -dsco NAME=DSCO -lco NAME=LCO \
  -sql 'SELECT NAME FROM "0.contours"'

<Document id="root_doc">
  <name>DSCO</name>
  <Document id="_0.contours">
    <name>LCO</name>
    <Placemark id="_0.contours.1">
      <name>-1500</name>
      <LineString>
```

7.42 Dutch Kadaster LV BAG 2.0 Extract

Added in version 3.2.

Driver short name

LVBAG

Build dependencies

libexpat

This driver can read XML files in the LV BAG 2.0 extract format as provided by the Dutch Kadaster BAG products. All LV BAG 2.0 extract products are supported. The driver supports all BAG layers including those introduced in BAG 2.0.

The driver is only available if GDAL/OGR is compiled against the Expat library.

Each extract XML file is presented as a single OGR layer. The layers are georeferenced in their native (EPSG:28992) SRS.

More information about the LV BAG 2.0 can be found at https://www.kadaster.nl/zakelijk/producten/adressen-en-gebouwen/bag-2.0-extract

LV BAG model definitions are available at https://developer.kadaster.nl/schemas/-/categories/4119958

Note 1 : The earlier BAG 1.0 extract is **not supported** by this driver.

Note 2 : The driver will only read ST (Standaard Levering) extract files. Mutation ML (Mutatie Levering) files are not supported.

7.42.1 Open options

The following open options can be specified (typically with the **-oo** name=value parameters of ogrinfo or ogr2ogr):

- **AUTOCORRECT_INVALID_DATA=[YES/NO]:** Defaults to `NO`. Whether or not the driver must try to adjust the data if a feature contains invalid or corrupted data. This typically includes fixing invalid geometries (with GEOS >= 3.8.0), dates, object status etc.
- **LEGACY_ID=[YES/NO]:** Defaults to `NO`. Format the BAG identifiers compatible with BAG 1.0.

7.42.2 VSI Virtual File System API support

The driver supports reading from files managed by VSI Virtual File System API, which include "regular" files, as well as files in the /vsizip/ domain. See examples below.

7.42.3 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.42.4 Examples

7.42.4.1 Example 1: Dumping the results of a LV BAG extract to WGS84 in GeoJSON

```
ogr2ogr -t_srs EPSG:4326 -f GeoJSON output.json 9999PND01012020_000001.xml
```

7.42.4.2 Example 2: Dumping contents of extract file as OGR sees it

```
ogrinfo -ro 9999PND01012020_000001.xml
```

7.42.4.3 Example 3: Inserting repaired features from LV BAG extract archive into PostgreSQL as WGS84 geometries

The table 'pand' will be created with the features from 9999PND18122019.zip. The database instance (lvbag) must already exist, and the table 'pand' must not already exist.

```
ogr2ogr -oo AUTOCORRECT_INVALID_DATA=YES -t_srs EPSG:4326 -f PostgreSQL PG:dbname=lvbag /
↪vsizip/9999PND18122019.zip
```

7.42.4.4 Example 4: Loading a LV BAG extract directory into Postgres

```
ogr2ogr \
  -f "PostgreSQL" PG:dbname="my_database" \
  9999PND18122019/ \
  -nln "name_of_new_table"
```

7.42.4.5 Example 5: Creating a GeoPackage from 'Nummeraanduiding' dataset

```
ogr2ogr \
  -f "GPKG" nummeraanduiding.gpkg \
  0000NUM01052020/
```

7.42.5 See Also

- Kadaster LV BAG 2.0 page (Dutch)

7.43 MapML

Added in version 3.1.

Driver short name

MapML

Driver built-in by default

This driver is built-in by default

This driver implements read and write support for the MapML specification. It only implements reading and writing vector features.

Warning

This driver implements an experimental specification, and inherits its experimental status. This specification may change at a later point, or not be adopted. Files written by this driver may no longer be readable in later versions of GDAL.

7.43.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.43.2 Read support

Layers are identified with the `class` attribute of features.

Fields are retrieved from the HTML table in the `properties` element of features. This assumes that they are written following the exact same structure as the write side of the driver. Otherwise no fields will be retrieved. Field type is guessed from the values, and may consequently be sometimes inaccurate.

7.43.3 Write support

Several layers can be written in the same MapML file.

Only the following CRS are natively supports, EPSG:4326 (WGS 84), EPSG:3857 (WebMercator), EPSG:3978 (NAD83 / Canada Atlas Lambert) and EPSG:5936 (WGS 84 / EPSG Alaska Polar Stereographic). Layers in other CRS will be automatically reprojected to EPSG:4326.

Geometry types Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon and GeometryCollection are supported.

Attributes are written as a HTML table.

7.43.4 Dataset creation options

Dataset creation options can be specified in command-line tools using the syntax `-dsco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following dataset creation options are supported:

- **HEAD=value:** Filename or inline XML content for head element.
- **EXTENT_UNITS=[AUTO/WGS84/OSMTILE/CBMTILE/APSTILE]:** To override the CRS.
- **EXTENT_XMIN=value:** Override extent xmin value.
- **EXTENT_YMIN=value:** Override extent ymin value.
- **EXTENT_XMAX=value:** Override extent xmax value.
- **EXTENT_YMAX=value:** Override extent ymax value.
- **EXTENT_ZOOM=<integer>:** Value of extent.zoom.
- **EXTENT_ZOOM_MIN=<integer>:** Min value for extent.zoom.
- **EXTENT_ZOOM_MAX=<integer>:** Max value for extent.zoom.
- **HEAD_LINKS=value:** Inline XML content for extra content to insert as link elements in the map-head. For example '<link rel="license" href="bar" title="foo" /><link type="baz" href="baw" />'

7.43.5 Links

- MapML.js documentation
- MapML specification
- MapML schemas
- *gdal2tiles* mapml output

7.44 MEM -- In Memory datasets

Added in version 3.11.

Driver short name

MEM

Driver built-in by default

This driver is built-in by default

GDAL supports the ability to hold datasets in a temporary in-memory format. This is primarily useful for temporary datasets in scripts or internal to applications. It is not generally of any use to application end-users.

This page documents its vector capabilities. Starting with GDAL 3.11, this driver has been unified with the long-time existing Memory driver to offer both *raster* and vector capabilities.

This driver implements read and write access layers of features contained entirely in memory. This is primarily useful as a high performance, and highly malleable working data store. All update options, geometry types, and field types are supported.

There is no way to open an existing vector MEM dataset. It must be created with Create(name, 0, 0, 0, GDT_Unknown) and populated and used from that handle. When the dataset is closed all contents are freed and destroyed.

The driver does not implement spatial or attribute indexing, so spatial and attribute queries are still evaluated against all features. Fetching features by feature id should be very fast (just an array lookup and feature copy).

7.44.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

7.44.2 Creation Issues

Any name may be used for a created dataset. There are no dataset creation options supported. Layer names need to be unique, but are not otherwise constrained.

Sparse IDs can be handled when using CreateFeature().

New fields can be added or removed to a layer that already has features, although the user is responsible for making sure that no `OGRFeature` instance is accessible outside of the driver before changing the layer definition.

7.44.2.1 Layer creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALDatasetCreateLayer()`* (C) or *`Dataset.CreateLayer`* (Python). The following layer creation options are available:

- **ADVERTIZE_UTF8=[YES/NO]:** Defaults to `NO`. Whether the layer will contain UTF-8 strings.
- **FID=<string>:** (GDAL >= 3.8) Defaults to `empty` string. Name of the FID column to create.

7.45 Memory (deprecated)

Driver short name

Memory

Driver built-in by default

This driver is built-in by default

Since 3.11, this driver is deprecated. Its functionality has been merged into the *MEM -- In Memory datasets* driver, and querying the `Memory` driver returns the new raster and vector capable `MEM` driver.

7.46 MiraMon Vector

Added in version 3.9.

Driver short name

MiraMonVector

Driver built-in by default

This driver is built-in by default

This driver is capable of translating (reading and writing) structured vectors of point, arc (*linestrings*), and polygon types from MiraMon vector format.

In MiraMon the concepts of OGRMultiPoints and OGRMultiLineStrings are not supported, but the driver translates a multipoint into N points and a multistring into N arcs. When reading a MiraMon file of type *.pol*, the corresponding layer will be reported as of type wkbPolygon, but depending on the number of parts of each geometry, the actual type of the geometry for each feature can be either OGRPolygon or OGRMultiPolygon.

The reading driver verifies if multipart polygons adhere to the specification (that is to say, the vertices of outer rings should be oriented clockwise on the X/Y plane, and those of inner rings counterclockwise). Otherwise, the driver corrects the orientation (in the original format this specification is not the case as polygon files are based on topological arc files, where the order of the vertices may be relevant).

Measures (M coordinate) are not supported. Symbolization is neither read nor generated by this driver.

A look-up-table of MiraMon and EPSG Spatial Reference Systems allows matching identifiers in both systems.

If a layer contains an old *.rel* format file (used some decades ago), a warning message will appear explaining how to convert it into a modern *.rel 4* file.

7.46.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.46.2 Overview of MiraMon format

The MiraMon format is a binary format for vector layer data, linked to one or more database tables, with or without topology and with rich metadata. More information about the structured MiraMon vector format is available on the public specification.

It is important to keep in mind that a MiraMon vector layer is composed by several files as follows:

To operate with a point layer, you must provide the name with the extension .pnt (the T.dbf and T.rel files must accompany the .pnt).

To operate with a linestring layer, you must provide the name with the extension .arc (the A.dbf and A.rel, .nod, N.dbf, and N.rel files must accompany the .arc).

To operate with a polygon layer, you must provide the name with the extension .pol (the P.dbf, P.rel, A.dbf and A.rel, .nod, N.dbf, and N.rel files must accompany the .pol).

By providing only the main file name, the driver will automatically use the other sidecar files to obtain the necessary information. In the creation of MiraMon layers, you only need to provide the name of the main file (with or without extension), and the driver will create the rest of the files.

The following outlines the information contained within each sidecar file:

Preliminary note: *FileName* is, in the following explanations, the first part of the name of the layer file.

- **Point layers**: These layers contain *point* type features which are described by a single coordinate (x,y) or (x, y, z). Each layer is composed by 3 files:
  - *FileName.pnt* file: Contains the geographic database with the coordinates that define the point vector features.
  - *FileNameT.dbf* file (note the 'T' before the '.'): Contains the main table of the database in dBASE (DBF) format, or in extended DBF format, if necessary. It contains the information (usually alphanumeric, but also file or web links, etc) of every feature. The Feature Identifier (FID) field is a field called *ID_GRAFIC* and relates every geographical feature to one or more records in the main table.
  - *FileNameT.rel* file (note the 'T' before the '.'): Contains the layer metadata, the relational structure of the database (links between the main table and other tables [thesauruses, etc] if needed, and the cardinality

of the link) and the default symbolization description. In the GDAL environment only some aspects are documented: the spatial reference system, the language of the metadata (English), the extension and a description of the fields.

- **Arc layers**: These layers contain *linestring* type features which are lines described by a series of segments, each one defined by coordinates (x, y) or (x, y, z). Both extreme vertices of each *linestring* are called nodes. Each layer is composed by 6 files:
  - *FileName.arc* file: Contains the geographic database with the coordinates that define the linestring (arc) vector features.
  - *FileNameA.dbf* file (note the 'A' before the '.'): Contains the main table of the database in dBASE (DBF) format, or in extended DBF format, if necessary. It contains the information (usually alphanumeric, but also file or web links, etc) of every feature. The Feature Identifier (FID) field is a field called *ID_GRAFIC* and relates every geographical feature to one or more records in the main table.
  - *FileNameA.rel* file (note the 'A' before the '.'): Contains the layer metadata, the relational structure of the database (links between the main table and other tables [thesauruses, etc] if needed, and the cardinality of the link) and the default symbolization description. In the GDAL environment only some aspects are documented: the spatial reference system, the language of the metadata (English), the extension and a description of the fields.
  - *FileName.nod* file: Contains the geographic database with information about the linestring needed to define each node. It is necessary in the MiraMon vector format but not read by the GDAL MiraMon vector driver because nodes contain topological information that is not transferred to other formats.
  - *FileNameN.dbf* file (note the 'N' before the '.'): Contains the main table of the database in dBASE (DBF) format, or in extended DBF if necessary. This table contains information about the relationships between arcs and nodes, and other attributes of the nodes, if needed. It is necessary in the MiraMon vector format but not read by the GDAL MiraMon vector driver because nodes contain topological information that is not transferred to other formats.
  - *FileNameN.rel* file (note the 'N' before the '.'): Contains the layer metadata, the relational structure of the database (links between the main table and other tables [thesauruses, etc] if needed, and the cardinality of the link) and the default symbolization description. It is necessary in the MiraMon vector format but not read by the GDAL MiraMon vector driver because nodes contain topological information that is not transferred to other formats.
- **Polygon layers**: These layers contain *polygon* or *multipolygon* type features. In MiraMon vector format a polygon is a closed shape described by one or more arcs. A polygon can have holes inside it. A polygon can also be linked to other polygons; in this case, it is termed a group (*multipolygon*). Each layer is composed by 9 files:
  - *FileName.pol* file: Contains the geographic database with information about the linestring vector features needed to define the polygon (or multipolygon) vector features.
  - *FileNameP.dbf* file (note the 'P' before the '.'): Contains the main table of the database in dBASE (DBF) format, or in extended DBF format, if necessary. It contains the information (usually alphanumeric, but also file or web links, etc) of every feature. The Feature Identifier (FID) field is a field called *ID_GRAFIC* and relates every geographical feature to one or more records in the main table.
  - *FileNameP.rel* file (note the 'P' before the '.'): Contains the layer metadata, the relational structure of the database (links between the main table and other tables [thesauruses, etc] if needed, and the cardinality of the link) and the default symbolization description. In the GDAL environment only some aspects are documented: the spatial reference system, the language of the metadata (English), the extension and a description of the fields.
  - *FileName.arc* file: Contains the geographic database with the coordinates that define the arc vector features. The polygons within the polygon file reference the arcs in this file by their index.

- *FileNameA.dbf* file (note the 'A' before the '.'): Contains the main table of the database in dBASE (DBF) format, or in extended DBF if necessary. This table contains information about the relationship between arcs and polygons, not the main features information. It is necessary in MiraMon but not read directly by the GDAL MiraMon vector driver because it is redundant to the information on the linestring part.
- *FileNameA.rel* file (note the 'A' before the '.'): Contains additional data about the data, the relations of the database and the symbolization description. It is necessary in MiraMon but not read directly by the GDAL MiraMon vector driver.
- *FileName.nod* file: Contains the geographic database with information about the linestring needed to define each node. It is necessary in the MiraMon vector format but not read by the GDAL MiraMon vector driver because nodes contain topological information that is not transferred to other formats.
- *FileNameN.dbf* file (note the 'N' before the '.'): Contains the main table of the database in dBASE (DBF) format, or in extended DBF if necessary. This table contains information about the relationships between arcs and nodes, and other attributes of the nodes, if needed. It is necessary in the MiraMon vector format but not read by the GDAL MiraMon vector driver because nodes contain topological information that is not transferred to other formats.
- *FileNameN.rel* file (note the 'N' before the '.'): Contains additional data about the data, the relations of the database and the symbolization description. It is necessary in MiraMon but not read directly by the GDAL MiraMon vector driver.

7.46.3 Encoding

When reading MiraMon files the code page setting in the header of the .dbf file is read and used to translate string fields to UTF-8 (regardless of whether they are in ANSI, OEM or UTF-8).

When writing MiraMon files the codepage of *.dbf* files can be ANSI or UTF8 depending on the layer creation option DBFEncoding.

7.46.4 Creation Issues

MiraMon can only store one kind of geometry per layer (points, arcs or polygons). Mixing different kinds of layers (including raster and geoservices as WMS or WMTS) is possible through MiraMon maps (.mmm). During creation the driver generates the necessary files to accommodate each of the three possible types of geometries. For instance, if a layer or a dataset contains points and arcs, a set of point files and a set of arc files will be created.

Consequently, during creation the MiraMon vector driver output can be a folder or a set of files with the appropriate extension (*.pnt*, etc):

- If the output is a **folder**, it will contain all the translated layers with the original name in the origin dataset.
  - In this case a *.mmm* file will be created referencing all layers in the origin dataset to make an easy open of the dataset using the MiraMon software.
  - In this case, please specify the MiraMon file output format name using the -f option (**-f MiraMonVector**).
- If the output is a **file** with extension all the translated layers in the origin dataset will be created with the specified name. Use this option only when you know that there is only one layer with one feature type in the origin dataset.

The attributes of the MiraMon feature are stored in an associated *.dbf*. If a classical DBF IV table could not be used (too many fields or records, large text fields, etc) a file type called extended DBF is used. This is an improvement of dBASE IV DBF files. The specification of this format can be found in this file.

Note that extended *.dbf* files cannot be opened with Excel or other typical programs. If the complete MiraMon Professional software is not installed on the computer, the free and standalone MiraD application can be downloaded from this page to open them.

7.46.5 Connection string

The MiraMon driver accepts three types of sources of data:

When translating from a MiraMon vector format, the MiraMon vector driver input needs a file with one of the described extensions:

- *.pnt* for *points*.
- *.arc* for *linestrings*.
- *.pol* for *polygons* (or *multipolygons*).

The extension *.nod* is not valid for translation. Take in consideration all auxiliary files described above.

7.46.6 Field sizes

The driver will automatically extend string and integer fields to dynamically accommodate the length of the data to be inserted.

7.46.7 Size Issues

Geometry: The MiraMon vector format explicitly uses 32-bit offsets in the 1.1 version and 64-bit offsets in the 2.0 version. It is better to produce 1.1 version files if 2.0 version is not really necessary than always use 2.0 version. Version 1.x files are smaller.

Attributes: The dbf format does not have any offsets in it, so it can be arbitrarily large.

7.46.8 Open options

The following open options are available.

- **Height=[First/Lowest/Highest]:** Sets which of the possible heights for each vertex is read: the *first*, the *lowest* or the *highest* one. It only applies to MiraMon multi-height layers, where the same X,Y vertex can have more than one Z.
- **MultiRecordIndex=[1/2/.../Last/JSON]:** In case of fields of type List, if the output driver cannot support them, user can select which one wants to keep: *MultiRecordIndex=1* for first, *MultiRecordIndex=2* for second, etc and *MultiRecordIndex=last* for the last element of the list. *MultiRecordIndex=JSON* option converts the list in a single value in JSON format. If not specified, all elements of the list will be translated by default as a OGR list field type.
- **OpenLanguage=[ENG/CAT/SPA]:** Defaults to `ENG`. If the layer to be opened is multilingual (in fact, the *.rel* file), this parameter sets the language to be read.

7.46.9 Dataset creation options

None

7.46.10 Layer creation options

- **Version=[V1.1/V2.0/last_version]:** Defaults to `V1.1`. Version of the file. Version 1.1 is limited to an unsigned 32-bit integer for FID, for internal offsets and for the number of entities the layer can handle. It is the default option. Version 2.0 is the 64-bit version. It is practically unlimited (unsigned 64-bit integer for FID and internal offsets). last_version selects to the last existing version ever.
- **DBFEncoding=[UTF8/ANSI]:** Defaults to `ANSI`. Encoding of the *.dbf* files. The MiraMon vector driver can write *.dbf* files in UTF-8 or ANSI charsets. As at the moment of this release, UTF-8 tables are not editable in the MiraD application, so it is recommended to use ANSI instead, if there are no coding problems.

- **CreationLanguage=[ENG/CAT/SPA]:** Defaults to ENG. Sets the language used in the metadata file (*.rel*) for the descriptors of the *.dbf* fields.

7.46.11 Examples

7.46.11.1 Example 1

A translation from an *Example_1.dxf* file with one layer but some different geometric types in the layer, will result 'file1.dxf' into a new MiraMon set of layers in the 'output_folder'.

```
ogr2ogr output_folder Example_1.dxf -f MiraMonVector -lco Version=V1.1
```

7.46.11.2 Example 2

A translation from a *Example_2.dxf* file with one polygon type layer 'file1.dxf' into a new MiraMon layer 'territories.pol' (with UTF-8 encoding at the *.dbf* files) is performed like this:

```
ogr2ogr territories.pol Example_2.dxf -lco DBFEncoding=UTF8 (no needed to include **-f␣
↪MiraMonVector** because the output layer is not a directory)
```

7.46.11.3 Example 3

A translation from a MiraMon layer of arcs, 'rivers.arc', into a new *.gml* file (taking only the first element of the multi-records in the attributes table) is performed like this:

```
ogr2ogr rivers.gml rivers.arc -oo MultiRecordIndex=1
```

7.46.11.4 Example 4

A translation from a MiraMon layer 'tracks.arc' into a new *.gml* file taking the first height of every point is performed like this:

```
ogr2ogr tracks.gml tracks.arc -oo Height=First
```

7.46.11.5 Example 5

A translation from a MiraMon layer 'tracks.arc' into a new *.gml* file taking the last height of every point and documenting the attribute descriptors in Catalan (if the layer is multilingual and it has this language available) is performed like this:

```
ogr2ogr tracks.gml tracks.arc -oo Height=First -oo Language=CAT
```

7.46.12 See Also

- MiraMon's vector format specifications
- MiraMon Extended DBF format
- MiraMon vector layer concepts.
- MiraMon page
- MiraMon help guide
- Grumets research group, the people behind MiraMon

7.47 MapInfo TAB and MIF/MID

Driver short name

MapInfo File

Driver built-in by default

This driver is built-in by default

MapInfo datasets in native (TAB) format and in interchange (MIF/MID) format are supported for reading and writing. Update of existing TAB files is supported (append of new features, modifications and deletions of existing features, adding/renaming/deleting fields, ...). Update of existing MIF/MID files is not supported.

Note: In the rest of this document "MIF/MID File" is used to refer to a pair of .MIF + .MID files, and "TAB file" refers to the set of files for a MapInfo table in binary form (usually with extensions .TAB, .DAT, .MAP, .ID, .IND).

The MapInfo driver treats a whole directory of files as a dataset, and a single file within that directory as a layer. In this case the directory name should be used as the dataset name.

However, it is also possible to use one of the files (.tab or .mif) in a MapInfo set as the dataset name, and then it will be treated as a dataset with one single layer.

MapInfo coordinate system information is supported for reading and writing.

7.47.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.47.2 Creation Issues

The TAB File format requires that the bounds (geographical extents) of a new file be set before writing the first feature.

There is currently no automated setting of valid default bounds for each spatial reference system, so for the time being, the MapInfo driver sets the following default bounds when a new layer is created:

- For a file in LAT/LON (geographic) coordinates: BOUNDS (-180, -90) (180, 90)
- For any other projection: BOUNDS (-30000000 + false_easting, -15000000 + false_northing) (30000000 + false_easting, 15000000 + false_northing)

It is possible to override those bounds through two mechanisms.

- specify a user-defined file that contain projection definitions with bounds. The name of this file must be specified with the *MITAB_BOUNDS_FILE* configuration option. This allows users to override the default bounds for existing projections, and to define bounds for new projections not listed in the hard-coded table in the driver. The format of the file is a simple text file with one CoordSys string per line. The CoordSys lines should follow the MIF specs, and MUST include the optional Bounds definition at the end of the line, e.g.

```
# Lambert 93 French bounds
CoordSys Earth Projection 3, 33, "m", 3, 46.5, 44, 49.00000000002, 700000, 6600000␣
↪Bounds (75000, 6000000) (1275000, 7200000)
```

  It is also possible to establish a mapping between a source CoordSys and a target CoordSys with bounds. Such a mapping is specified by adding a line starting with "Source = " followed by a CoordSys (spaces before or after the equal sign do not matter). The following line should start with "Destination = " followed by a CoordSys with bounds, e.g.

```
# Map generic Lambert 93 to French Lambert 93, Europe bounds
Source      = CoordSys Earth Projection 3, 33, "m", 3, 46.5, 44, 49, 700000, 6600000
Destination = CoordSys Earth Projection 3, 33, "m", 3, 46.5, 44, 49.00000000001,␣
↪700000, 6600000 Bounds (-792421, 5278231) (3520778, 9741029)
```

- use the *BOUNDS* layer creation option (see below)

If no coordinate system is provided when creating a layer, the projection case is used, not geographic, which can result in very low precision if the coordinates really are geographic. You can add "-a_srs WGS84" to the **ogr2ogr** commandline during a translation to force geographic mode.

MapInfo feature attributes suffer a number of limitations:

- Only Integer, Real and String field types can be created. The various list, and binary field types cannot be created.
- For String fields, the field width is used to establish storage size in the .dat file. This means that strings longer than the field width will be truncated
- String fields without an assigned width are treated as 254 characters.

7.47.2.1 Dataset Creation Options

Dataset creation options can be specified in command-line tools using the syntax `-dsco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following dataset creation options are supported:

- **FORMAT=[TAB/MIF]:** To create MIF/MID instead of TAB files
- **SPATIAL_INDEX_MODE=[QUICK/OPTIMIZED]:** Defaults to `QUICK`. In QUICK mode writing files can be about 5 times faster, but spatial queries can be up to 30 times slower. This can be set to OPTIMIZED to generate optimized spatial index.
- **BLOCKSIZE=[512/1024/.../32256]:** Defaults to `512`. (multiples of 512) Block size for .map files. MapInfo 15.2 and above creates .tab files with a blocksize of 16384 bytes. Any MapInfo version should be able to handle block sizes from 512 to 32256.
- **STRICT_FIELDS_NAME_LAUNDERING=[YES/NO]:** (GDAL >= 3.10) Defaults to `YES`. Replaces all non alphanumeric characters in dataset's field names by _ (underscore). For recent MapInfo can be set to *NO*.

7.47.2.2 Layer Creation Options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALDatasetCreateLayer()* (C) or *Dataset.CreateLayer* (Python). The following layer creation options are supported:

- **BOUNDS=<xmin,ymin,xmax,ymax>:** Define custom layer bounds to increase the accuracy of the coordinates. Note: the geometry of written features must be within the defined box.
- **ENCODING=value:** Define the encoding for field names and field values. The encoding name is specified in the format supported by *CPLRecode()* (e.g. ISO-8859-1, CP1251, CP1252 ...) and internally converted to MapInfo charsets names. Default value is '' (empty string) that equals to 'Neutral' MapInfo charset. Currently supported values for the encoding name are:

Table 4: MapInfo encodings

ENCODING value	MapInfo charset	description
"" (empty string)	Neutral	No character conversions performed.
UTF-8	UTF-8	UTF-8 (Works with recent MapInfo versions, since v15.2)
ISO-8859-1	ISO8859_1	ISO 8859-1 (UNIX)
ISO-8859-2	ISO8859_2	ISO 8859-2 (UNIX)
ISO-8859-3	ISO8859_3	ISO 8859-3 (UNIX)
ISO-8859-4	ISO8859_4	ISO 8859-4 (UNIX)
ISO-8859-5	ISO8859_5	ISO 8859-5 (UNIX)
ISO-8859-6	ISO8859_6	ISO 8859-6 (UNIX)
ISO-8859-7	ISO8859_7	ISO 8859-7 (UNIX)
ISO-8859-8	ISO8859_8	ISO 8859-8 (UNIX)
ISO-8859-9	ISO8859_9	ISO 8859-9 (UNIX)
EUC-JP	PackedEUCJapaese	UNIX, standard Japanese implementation.
CP1252	WindowsLatin1	
CP1250	WindowsLatin2	
CP1256	WindowsArabic	
CP1251	WindowsCyrillic	
CP1257	WindowsBalticRim	
CP1253	WindowsGreek	
CP1255	WindowsHebrew	
CP1254	WindowsTurkish	Windows Eastern Europe
CP950	WindowsTradChinese	Windows Traditional Chinese
CP936	WindowsSimpChinese	Windows Simplified Chinese
CP932	WindowsJapanese	
CP949	WindowsKorean	
CP437	CodePage437	DOS Code Page 437 = IBM Extended ASCII
CP850	CodePage850	DOS Code Page 850 = Multilingual
CP852	CodePage852	DOS Code Page 852 = Eastern Europe
CP855	CodePage855	DOS Code Page 855 = Cyrillic
CP857	CodePage857	
CP860	CodePage860	DOS Code Page 860 = Portuguese
CP861	CodePage861	DOS Code Page 861 = Icelandic
CP863	CodePage863	DOS Code Page 863 = French Canadian
CP864	CodePage864	DOS Code Page 864 = Arabic
CP865	CodePage865	DOS Code Page 865 = Nordic
CP869	CodePage869	DOS Code Page 869 = Modern Greek
(no iconv match)	LICS	Lotus worksheet release 1,2 character set
(no iconv match)	LMBCS	Lotus worksheet release 3,4 character set

- **DESCRIPTION=value:** (GDAL >= 3.1.0) Friendly layer name (only for TAB format). Friendly names can be up to 256 characters long and can include most ASCII characters. Supported by MapInfo Pro v15.0 or higher.
- **STRICT_FIELDS_NAME_LAUNDERING=[YES/NO]:** (GDAL >= 3.10) Defaults to `YES`. Replaces all non alphanumeric characters in layer's field names by _ (underscore). For recent MapInfo can be set to *NO*.

7.47.2.3 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The following configuration options are available:

- **MITAB_BOUNDS_FILE=value:** See *Creation Issues*.
- **MITAB_SET_TOWGS84_ON_KNOWN_DATUM=[YES/NO]:** (GDAL >= 3.0.3) The default behavior, starting with GDAL 3.0.3, is NO. That is, the TOWGS84 parameters read from the .tab header will *not* be set on the Datum object of the CRS, when the datum can be inferred.

7.47.2.4 See Also

- MITAB Page
- About friendly layer names

7.48 MongoDBv3

Added in version 3.0.

Driver short name

MongoDBv3

Build dependencies

Mongo CXX >= 3.4.0 client library

This driver can connect to the a MongoDB service.

The driver supports read, creation, update and delete operations of documents/features and collections/layers. The MongoDB database must exist before operating on it with OGR.

This driver requires the MongoDB C++ v3.4.0 client library.

7.48.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.48.2 MongoDB vs OGR concepts

A MongoDB collection within a database is considered as a OGR layer. A MongoDB document is considered as a OGR feature.

7.48.3 Dataset name syntax

There are two main possible syntaxes:

- One using MongoDB URI prefixed with *MONGODBV3:*, such as MONGODBV3:mongodb://[usr:pwd@]host1[:port1]...[,hostN[:portN]]][/[db][?options]]
- One using just MongoDBv3: as the name and open options to specify host, port, user, password, database, etc...

Note: the MONGODBV3: prefix before a URI starting with *mongodb://* is required to make it recognize by this driver, instead of the legacy driver. If the URI is starting with *mongodb+srv://*, then it is not needed.

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The open options available are:

- **URI=value:** Connection URI
- **HOST=value:** Defaults to `localhost`. Server hostname.
- **PORT=value:** Defaults to `27017`. Server port.
- **DBNAME=value:** Database name. Should be specified when connecting to hosts with user authentication enabled.
- **USER=value:** User name.
- **PASSWORD=value:** User password.
- **SSL_PEM_KEY_FILE=<filename>:** SSL PEM certificate/key filename.
- **SSL_PEM_KEY_PASSWORD=value:** SSL PEM key password.
- **SSL_CA_FILE=<filename>:** SSL Certification Authority filename.
- **SSL_CRL_FILE=<filename>:** SSL Certification Revocation List filename.
- **SSL_ALLOW_INVALID_CERTIFICATES=[YES/NO]:** Defaults to `NO`. Whether to allow connections to servers with invalid certificates.
- **BATCH_SIZE=value:** Number of features to retrieve per batch. For most queries, the first batch returns 101 documents or just enough documents to exceed 1 megabyte. Subsequent batch size is 4 megabytes.
- **FEATURE_COUNT_TO_ESTABLISH_FEATURE_DEFN=value:** Defaults to `100`. Number of features to retrieve to establish feature definition. -1 = unlimited.
- **JSON_FIELD=[YES/NO]:** Defaults to `NO`. Whether to include a field called "_json" with the full document as JSON.
- **FLATTEN_NESTED_ATTRIBUTE=[YES/NO]:** Defaults to `YES`. Whether to recursively explore nested objects and produce flatten OGR attributes.
- **FID=value:** Defaults to `ogc_fid`. Field name, with integer values, to use as FID.
- **USE_OGR_METADATA=[YES/NO]:** Defaults to `YES`. Whether to use the _ogr_metadata collection to read layer metadata.

- **BULK_INSERT=[YES/NO]:** Defaults to `YES`. Whether to use bulk insert for feature creation.

7.48.4 Filtering

The driver will forward any spatial filter set with SetSpatialFilter() to the server when a "2d" or "2dsphere" spatial index is available on the geometry field.

However, in the current state, SQL attribute filters set with SetAttributeFilter() are evaluated only on client-side. To enable server-side filtering, the string passed to SetAttributeFilter() must be a JSON object in the MongoDB filter syntax.

7.48.5 Paging

Features are retrieved from the server by chunks of 101 documents or just enough documents to exceed 1 megabyte. Subsequent batch size is 4 megabytes. This can be altered with the BATCH_SIZE open option.

7.48.6 Schema

When reading a MongoDB collection, OGR must establish the schema of attribute and geometry fields, since, contrary to MongoDB collections which are schema-less, OGR has a fixed schema concept.

In the general case, OGR will read the first 100 documents (can be altered with the FEATURE_COUNT_TO_ESTABLISH_FEATURE_DEFN open option) of the collection and build the schema that best fit to the found fields and values.

If the collection/layer has been previously created with OGR, a _ogr_metadata special collection contains the OGR schema, in which case it will be directly used. It might be possible to ignore the schema written in _ogr_metadata by setting the USE_OGR_METADATA=NO open option.

It is also possible to set the JSON_FIELD=YES open option so that a _json special field is added to the OGR schema. When reading MongoDB documents as OGR features, the full JSON version of the document will be stored in the _json field. This might be useful in case of complex documents or with data types that do not translate well in OGR data types. On creation/update of documents, if the _json field is present and set, its content will be used directly (other fields will be ignored).

7.48.7 Feature ID

MongoDB have a special _id field that contains the unique ID of the document. This field is returned as an OGR field, but cannot be used as the OGR special FeatureID field, which must be of integer type. By default, OGR will try to read a potential 'ogc_fid' field to set the OGR FeatureID. The name of this field to look up can be set with the FID open option. If the field is not found, the FID returned by OGR will be a sequential number starting at 1, but it is not guaranteed to be stable at all.

7.48.8 ExecuteSQL() interface

If specifying "MongoDB" as the dialect of ExecuteSQL(), a JSON string with a serialized MongoDB command can be passed. The result will be returned as a JSON string in a single OGR feature.

Standard SQL requests will be executed on client-side.

7.48.9 Write support

Layer/collection creation and deletion is possible.

Write support is only enabled when the datasource is opened in update mode.

When inserting a new feature with CreateFeature(), and if the command is successful, OGR will fetch the returned _id and use it for the SetFeature() operation.

7.48.10 Layer creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALDatasetCreateLayer()* (C) or *Dataset.CreateLayer* (Python). The following layer creation options are supported:

- **OVERWRITE=[YES/NO]:** Defaults to `NO`. Whether to overwrite an existing collection with the layer name to be created.
- **GEOMETRY_NAME=value:** Defaults to `geometry`. Name of geometry column.
- **SPATIAL_INDEX=[YES/NO]:** Defaults to `YES`. Whether to create a spatial index (2dsphere).
- **FID=value:** Defaults to `ogc_fid`. Field name, with integer values, to use as FID.
- **WRITE_OGR_METADATA=[YES/NO]:** Defaults to `YES`. Whether to create a description of layer fields in the _ogr_metadata collection.
- **DOT_AS_NESTED_FIELD=[YES/NO]:** Defaults to `YES`. Whether to consider dot character in field name as sub-document.
- **IGNORE_SOURCE_ID=[YES/NO]:** Defaults to `NO`. Whether to ignore _id field in features passed to CreateFeature().

7.48.11 Examples

7.48.11.1 Example 1: Listing the tables of a MongoDB database

```
ogrinfo -ro mongodb+srv://user:password@cluster0-ox9uy.mongodb.net/test
```

7.48.11.2 Example 2: Filtering on a MongoDB field

```
ogrinfo -ro mongodb+srv://user:password@cluster0-ox9uy.mongodb.net/test -where '{ "field
↪": 5 }'
```

7.48.11.3 Example 3: Creating and populating a collection from a shapefile

```
ogr2ogr -update mongodb+srv://user:password@cluster0-ox9uy.mongodb.net/test shapefile.shp
```

7.48.12 Build instructions

GDAL/OGR must be built against the MongoDB C++ driver client library, v3.4.0, in order to the MongoDBv3 driver to be compiled.

You must first follow MongoDB C++ driver client build instructions.

Then:

- On Linux/Unix, run ./configure --with-mongocxxv3 (potentially by overriding PKG_CONFIG_PATH to point to the {INSTALLATION_PREFIX_OF_MONGOCXX}/lib/pkgconfig
- On Windows, uncomment and adapt the following in nmake.opt (or add in nmake.local):

```
# Uncomment for MongoDBv3 support
# Uncomment following line if plugin is preferred
#MONGODBV3_PLUGIN = YES
BOOST_INC=E:/boost_1_69_0
MONGOCXXV3_CFLAGS = -IE:/dev/install-mongocxx-3.4.0/include/mongocxx/v_noabi -IE:/
```

(continues on next page)

(continued from previous page)

```
↪dev/install-mongocxx-3.4.0/include/bsoncxx/v_noabi
MONGOCXXV3_LIBS = E:/dev/install-mongocxx-3.4.0/lib/mongocxx.lib E:/dev/install-
↪mongocxx-3.4.0/lib/bsoncxx.lib
```

7.48.13 See Also

- MongoDB C++ Driver
- MongoDB Manual

7.49 MSSQLSpatial - Microsoft SQL Server Spatial Database

Driver short name

MSSQLSpatial

Build dependencies

ODBC library

This driver implements support for access to spatial tables in Microsoft SQL Server 2008+ which contains the geometry and geography data types to represent the geometry columns.

7.49.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

7.49.2 Connecting to a database

To connect to a MSSQL datasource, use a connection string specifying the database name, with additional parameters as necessary. The connection strings must be prefixed with '*MSSQL:*'.

```
MSSQL:server=.\MSSQLSERVER2008;database=dbname;trusted_connection=yes
```

In addition to the standard parameters of the ODBC driver connection string format the following custom parameters can also be used in the following syntax:

- **Tables=schema1.table1(geometry column1),schema2.table2(geometry column2)**: By using this parameter you can specify the subset of the layers to be used by the driver. If this parameter is not set, the layers are retrieved from the geometry_columns metadata table. You can omit specifying the schema and the geometry column portions of the syntax.

- **GeometryFormat=native|wkb|wkt|wkbzm**: The desired format in which the geometries should be retrieved from the server. The default value is 'native' in this case the native SqlGeometry and SqlGeography serialization format is used. When using the 'wkb' or 'wkt' setting the geometry representation is converted to 'Well Known Binary' and 'Well Known Text' at the server. This conversion requires a significant overhead at the server and makes the feature access slower than using the native format. The wkbzm format can only be used with SQL Server 2012.

The parameter names are not case sensitive in the connection strings.

Specifying the **Database** parameter is required by the driver in order to select the proper database.

The connection may contain the optional **Driver** parameter if a custom SQL server driver should be loaded (like FreeTDS). The default is **{SQL Server}**.

Authentication is supported either through a **trusted_conection** or through username (**UID**) and password (**PWD**). As providing username and password on the commandline can be a security issue, login credentials can also be more securely stored in user defined environment variables *MSSQLSPATIAL_UID* and *MSSQLSPATIAL_PWD*.

7.49.3 Layers

If the user defines the environment variable `MSSQLSPATIAL_LIST_ALL_TABLES=YES` (and does not specify Tables= in the connection string), all regular user tables and views will be treated as layers. This option is useful if you want tables with with no spatial data

By default the MSSQL driver will only look for layers that are registered in the *geometry_columns* metadata table. If the user defines the environment variable `MSSQLSPATIAL_USE_GEOMETRY_COLUMNS=NO` then the driver will look for all user spatial tables and views found in the system catalog.

When `MSSQLSPATIAL_USE_GEOMETRY_COLUMNS=YES` (the default) is enabled, any datasets written by GDAL to the database are registered in the `dbo.geometry_columns` table.

This has the advantage of storing the dataset's coordinate reference system (CRS) alongside its metadata. GDAL will automatically use this information when the dataset is accessed in the future.

7.49.4 SQL statements

The MS SQL Spatial driver passes SQL statements directly to MS SQL by default, rather than evaluating them internally when using the ExecuteSQL() call on the OGRDataSource, or the -sql command option to ogr2ogr. Attribute query expressions are also passed directly through to MSSQL. It's also possible to request the OGR MSSQL driver to handle SQL commands with the *OGR SQL* engine, by passing **"OGRSQL"** string to the ExecuteSQL() method, as the name of the SQL dialect.

The MSSQL driver in OGR supports the OGRLayer::StartTransaction(), OGRLayer::CommitTransaction() and OGRLayer::RollbackTransaction() calls in the normal SQL sense.

7.49.5 Creation Issues

This driver doesn't support creating new databases, you might want to use the *Microsoft SQL Server Client Tools* for this purpose, but it does allow creation of new layers within an existing database.

7.49.5.1 Layer Creation Options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to `GDALDatasetCreateLayer()` (C) or `Dataset.CreateLayer` (Python). The following layer creation options are supported:

- **GEOM_TYPE=[geometry/geography]:** Defaults to `geometry`. The GEOM_TYPE layer creation option can be set to one of "geometry" or "geography". If this option is not specified the default value is "geometry". So as to create the geometry column with "geography" type, this parameter should be set "geography". In this

case the layer must have a valid spatial reference of one of the geography coordinate systems defined in the **sys.spatial_reference_systems** SQL Server metadata table. Projected coordinate systems are not supported in this case.

- **OVERWRITE=[YES/NO]:** This may be "YES" to force an existing layer of the desired name to be destroyed before creating the requested layer.
- **LAUNDER=[YES/NO]:** Defaults to `YES`. This may be "YES" to force new fields created on this layer to have their field names "laundered" into a form more compatible with MSSQL. This converts to lower case and converts some special characters like "-" and "#" to "_". If "NO" exact names are preserved. If enabled the table (layer) name will also be laundered.
- **PRECISION=[YES/NO]:** Defaults to `YES`. This may be "YES" to force new fields created on this layer to try and represent the width and precision information, if available using numeric(width,precision) or char(width) types. If "NO" then the types float, int and varchar will be used instead.
- **DIM=[2/3]:** Defaults to `3`. Control the dimension of the layer.
- **GEOMETRY_NAME=value:** Defaults to `ogr_geometry`. Set the name of geometry column in the new table.
- **SCHEMA=value:** Set name of schema for new table. If this parameter is not supported the default schema "*dbo*" is used.
- **SRID=value:** Set the spatial reference id of the new table explicitly. The corresponding entry should already be added to the spatial_ref_sys metadata table. If this parameter is not set the SRID is derived from the authority code of source layer SRS.
- **SPATIAL_INDEX=[YES/NO]:** Defaults to `YES`. Boolean flag to enable/disable the automatic creation of a spatial index on the newly created layers.
- **UPLOAD_GEOM_FORMAT=[wkb/wkt]:** Specify the geometry format (wkb or wkt) when creating or modifying features.
- **FID=ogr_fid:** Name of the FID column to create.
- **FID64=[YES/NO]:** Defaults to `NO`. Specifies whether to create the FID column with bigint type to handle 64bit wide ids.
- **GEOMETRY_NULLABLE=[YES/NO]:** Defaults to `YES`. Specifies whether the values of the geometry column can be NULL.
- **EXTRACT_SCHEMA_FROM_LAYER_NAME=[YES/NO]:** Defaults to `YES`. Can be set to NO to avoid considering the dot character as the separator between the schema and the table name.

7.49.6 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The following configuration options are available:

- **MSSQLSPATIAL_USE_BCP=[YES/NO]:** Enables bulk insert (BCP) mode when adding features. This option requires GDAL to be compiled against a BCP-enabled ODBC driver, such as ODBC Driver 18 for SQL Server.

  To explicitly specify a BCP-capable driver in the connection string, use the `DRIVER` parameter, for example:

  ```
  DRIVER={ODBC Driver 18 for SQL Server}
  ```

  If GDAL is compiled against SQL Server Native Client 10.0 or 11.0, the driver is selected automatically and does not need to be specified in the connection string.

  In these cases, the default value of this option is `YES`. Otherwise, the parameter is ignored by the driver.

- **MSSQLSPATIAL_BCP_SIZE=value:** Defaults to `1000`. Specifies the bulk insert batch size. The larger value makes the insert faster, but consumes more memory.
- **MSSQLSPATIAL_OGR_FID=value:** Defaults to `ogr_fid`. Override FID column name.
- **MSSQLSPATIAL_ALWAYS_OUTPUT_FID=[YES/NO]:** Defaults to `NO`. Always retrieve the FID value of the recently created feature (even if it is not a true IDENTITY column).
- **MSSQLSPATIAL_SHOW_FID_COLUMN=[YES/NO]:** Defaults to `NO`. Force to display the FID columns as a feature attribute.
- **MSSQLSPATIAL_USE_GEOMETRY_COLUMNS=[YES/NO]:** Defaults to `YES`. Use/create geometry_columns metadata table in the database.
- **MSSQLSPATIAL_LIST_ALL_TABLES=[YES/NO]:** Defaults to `NO`. Use mssql catalog to list available layers.
- **MSSQLSPATIAL_USE_GEOMETRY_VALIDATION=[YES/NO]:** (GDAL >= 3.0) Let the driver detect the geometries which would trigger run time errors at MSSQL server. The driver tries to correct these geometries before submitting that to the server.

7.49.7 Transaction support

The driver implements transactions at the dataset level, per rfc-54

7.49.8 Examples

7.49.8.1 Example 1: Creating a layer from an OGR data source

```
ogr2ogr -overwrite -f MSSQLSpatial "MSSQL:server=.\MSSQLSERVER2008;database=geodb;
↪trusted_connection=yes" "rivers.tab"

ogr2ogr -overwrite -f MSSQLSpatial "MSSQL:server=127.0.0.1;database=TestDB;UID=SA;
↪PWD=DummyPassw0rd" "rivers.gpkg"
```

7.49.8.2 Example 2: Connecting to a layer and dumping the contents

```
ogrinfo -al "MSSQL:server=.\MSSQLSERVER2008;database=geodb;tables=rivers;trusted_
↪connection=yes"

ogrinfo -al "MSSQL:server=127.0.0.1;database=TestDB;driver=ODBC Driver 17 for SQL Server;
↪UID=SA;PWD=DummyPassw0rd"
```

7.49.8.3 Example 3: Connecting with username/password

```
ogrinfo -al   MSSQL:server=.\MSSQLSERVER2008;database=geodb;trusted_connection=no;
↪UID=user;PWD=pwd
```

7.49.8.4 Example 4: Creating a layer from an OGR data source using the CLI

The user account must have permissions to create a new table in database.

Bash

PowerShell

```
conn="MSSQL:DRIVER={ODBC Driver 18 for SQL Server};SERVER=SQL22.mydomain.local;
↪DATABASE=geodb;uid=user;pwd=pwd;TrustServerCertificate=yes;"
gdal vector convert in.geojson $conn --overwrite-layer
```

```
$conn="MSSQL:DRIVER={ODBC Driver 18 for SQL Server};SERVER=SQL22.mydomain.local;
↪DATABASE=geodb;uid=user;pwd=pwd;TrustServerCertificate=yes;"
gdal vector convert in.geojson $conn --overwrite-layer
```

7.49.8.5 Example 5: Connecting with username/password stored in environment variables

The Microsoft SQL Server ODBC driver used in this example is available for both Windows and Linux platforms.

Bash

PowerShell

```
export MSSQLSPATIAL_UID=user
export MSSQLSPATIAL_PWD=pwd
gdal vector info "MSSQL:DRIVER={ODBC Driver 18 for SQL Server};SERVER=SQL22;
↪DATABASE=geodb;TrustServerCertificate=yes;" --summary
```

```
$env:MSSQLSPATIAL_UID="user"
$env:MSSQLSPATIAL_PWD="pwd"
gdal vector info "MSSQL:DRIVER={ODBC Driver 18 for SQL Server};SERVER=SQL22;
↪DATABASE=geodb;TrustServerCertificate=yes;" --summary
```

7.49.8.6 Example 6: Selecting from a database table and writing to a GeoPackage file

> In this example, the `geo.rivers` table is not registered in the `dbo.geometry_columns` table. As a result, the CRS must be specified manually to ensure it is correctly applied in the output dataset.

Bash

PowerShell

```
conn="MSSQL:DRIVER={ODBC Driver 18 for SQL Server};SERVER=SQL22.mydomain.local;
↪DATABASE=geodb;uid=user;pwd=pwd;TrustServerCertificate=yes;"
gdal vector pipeline \
    ! read "$conn" \
    ! sql --sql "SELECT oid, geom FROM geo.rivers WHERE hydroarea=10" \
    ! edit --crs=EPSG:2157 --output-layer "lyr" \
    ! write out.gpkg --overwrite
```

```
$conn="MSSQL:DRIVER={ODBC Driver 18 for SQL Server};SERVER=SQL22.mydomain.local;
↪DATABASE=geodb;uid=user;pwd=pwd;TrustServerCertificate=yes;"
gdal vector pipeline `
    ! read $conn `
    ! sql --sql "SELECT oid, geom FROM geo.rivers WHERE hydroarea=10" `
    ! edit --crs=EPSG:2157 --output-layer "lyr" `
    ! write out.gpkg --overwrite
```

7.49.8.7 Example 7: Check if bulk insert (BCP) mode is available

BCP support is available if `(BCP)` is shown in the driver capabilities, for example:

```
MSSQLSpatial -vector- (rw+u): Microsoft SQL Server Spatial Database (BCP)
```

Bash

PowerShell

```bash
gdal vector --formats | grep MSSQL
```

```powershell
gdal vector --formats | Select-String "MSSQL"
```

7.50 MVT: Mapbox Vector Tiles

Driver short name

MVT

Build dependencies

(requires SQLite and GEOS for write support)

The MVT driver can read and write Mapbox Vector Tile files, as standalone files, uncompressed or gzip-compressed (typical extensions are .pbf, .mvt, .mvt.gz), or a tileset at a given zoom level of such files. Write support requires GDAL to be built with libsqlite3 and GEOS support.

Mapbox Vector Tiles stored within a SQLite container conforming to the MBTiles format are handled by the *MBTiles* driver.

Tilesets of MVT files can for example be generated by tippecanoe or tileserver-gl. The output file hierarchy will contain a *metadata.json* file at its root and {z}/{x}/{y}.pbf files with the tiles, where z is the zoom level and (x, y) the coordinate of the tile in a given zoom level. The origin of the tiling system is the top-left tile (XYZ convention). For example, for zoom levels 0 and 1 :

```
/metadata.json
/0/
   0/
     0.pbf
/1/
   0/
     0.pbf
     1.pbf
   1/
     0.pbf
     1.pbf
```

The driver will assume by default EPSG:3857 (WebMercator) spatial reference system and Z/X/Y tiling structure, if opening a filename with {Z}/{X}/{Y}.pbf or {Z}-{X}-{Y}.pbf name, or a zoom level of a tileset. Otherwise integer coordinates will be reported.

Note: When opening a zoom level of a tileset, the driver will make no effort of stitching together geometries for features that overlap several tiles.

7.50.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.50.2 Connection strings

The following connection strings are supported:

- */path/to/some.pbf*
- *MVT:http[s]://path/to/some.pbf*
- */path/to/{Z}*: where {Z} is a zoom level between 0 and 30, tiles are in /path/to/{Z}/{X}/{Y}.pbf files, and /path/to/metadata.json or /path/to.json should typically exist.
- *MVT:http[s]://path/to/{Z]*

The MVT: prefix may be added before the filename or directory name to help forcing identification of datasets in some rare cases where non-guided identification would fail.

7.50.3 metadata.json

This file is typically generated by tippecanoe and has the following content:

```
{
    "name": "my_layername",
    "description": "my_layername",
    "version": "2",
    "minzoom": "0",
    "maxzoom": "0",
    "center": "2.500000,49.500000,0",
    "bounds": "2.000000,49.000000,3.000000,50.000000",
    "type": "overlay",
    "format": "pbf",
    "json": "{
        \"vector_layers\": [ {
            \"id\": \"my_layername\",
            \"description\": \"\",
            \"minzoom\": 0,
```

(continues on next page)

(continued from previous page)

```
            \"maxzoom\": 0,
            \"fields\": {
                \"bool_false\": \"Boolean\",
                \"bool_true\": \"Boolean\",
                \"float_value\": \"Number\",
                \"pos_int_value\": \"Number\",
                \"pos_int64_value\": \"Number\",
                \"neg_int_value\": \"Number\",
                \"neg_int64_value\": \"Number\",
                \"pos_sint_value\": \"Number\",
                \"pos_sint64_value\": \"Number\",
                \"neg_sint_value\": \"Number\",
                \"neg_sint64_value\": \"Number\",
                \"real_value\": \"Number\",
                \"string_value\": \"String\",
                \"uint_value\": \"Number\",
                \"uint64_value\": \"Number\"
            }
        } ],
        \"tilestats\": {
            \"layerCount\": 1,
            \"layers\": [
                {
                \"layer\": \"my_layername\",
                \"count\": 2,
                \"geometry\": \"Point\",
                \"attributeCount\": 0,
                \"attributes\": []
                }
            ]
        }
    }}"
}
```

The MVT driver only uses the "json" key to retrieve the layer names, their fields and the geometry type, and the "bounds" key for the layer extent.

If this file cannot be found, the layer schema is established by scanning the features of the tile(s).

As an extension, OGR handles in reading and writing custom tiling schemes by using the *crs*, *tile_origin_upper_left_x*, *tile_origin_upper_left_y*, *tile_dimension_zoom_0*, *tile_matrix_width_zoom_0* and *tile_matrix_height_zoom_0* metadata items. For example, for the Finnish ETRS-TM35FIN (EPSG:3067) tiling scheme:

```
{
  "...": "...",
  "crs":"EPSG:3067",
  "tile_origin_upper_left_x":-548576.0,
  "tile_origin_upper_left_y":8388608.0,
  "tile_dimension_zoom_0":2097152.0,
}
```

Or for a `WorldCRS84Quad` tiling scheme with 2 tiles in the horizontal direction at zoom level 0:

```
{
  "...": "...",
  "crs":"EPSG:4326",
  "tile_origin_upper_left_x":-180.0,
  "tile_origin_upper_left_y":90.0,
  "tile_dimension_zoom_0":180.0,
  "tile_matrix_width_zoom_0":2,
  "tile_matrix_height_zoom_0":1
}
```

7.50.4 Opening options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are available:

- **X=<integer>:** X coordinate of the EPSG:3857 tile.
- **Y=<integer>:** Y coordinate of the EPSG:3857 tile.
- **Z=<integer>:** Z coordinate of the EPSG:3857 tile.
- **METADATA_FILE=<filename>:** Filename of a metadata.json-like file. If opening a /path/to/{Z}/{X}/{Y}.pbf file, the driver will by default try to find /path/to/metadata.json. Setting the value to the empty string is a way of avoid the metadata.json file to be used.
- **CLIP=[YES/NO]:** Defaults to `YES`. Whether to clip geometries of vector features to tile extent. Generators of vector tiles will typically create geometries with a small buffer beyond the tile extent so that geometries intersecting several tiles can be unioned back. Defaults to YES so that that buffer is removed and geometries are clipped exactly to the tile extent.
- **TILE_EXTENSION=value:** Defaults to `pbf`. For tilesets, extension of tiles.
- **TILE_COUNT_TO_ESTABLISH_FEATURE_DEFN=<integer>:** Defaults to `1000`. For tilesets without metadata file, maximum number of tiles to use to establish the layer schemas.

  ADD_TILE_FIELDS=[YES/NO]: (GDAL >= 3.13) Defaults to `NO`. Whether to add fields 'tile_z', 'tile_x', 'tile_y' to each layer, containing the Z/X/Y coordinates of the tile from which the feature originates.

7.50.5 Creation issues

Tiles are generated with WebMercator (EPSG:3857) projection by default (custom tiling schemes can be defined with the TILING_SCHEME option). Several layers can be written. It is possible to decide at which zoom level ranges a given layer is written.

Part of the conversion is multi-threaded by default, using as many threads as there are cores. The number of threads used can be controlled with the *GDAL_NUM_THREADS* configuration option.

7.50.6 Dataset creation options

Dataset creation options can be specified in command-line tools using the syntax `-dsco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following dataset creation options are supported:

- **NAME=value:** Tileset name. Defaults to the basename of the output file/directory. Used to fill metadata records.
- **DESCRIPTION=value:** A description of the tileset. Used to fill metadata records.
- **TYPE=[overlay/baselayer]:** Layer type. Used to fill metadata records.

- **FORMAT=[DIRECTORY/MBTILES]:** Format into which tiles are written. DIRECTORY means that tiles are written in a hierarchy like out_dir/{z}/{x}/{y}.pbf. MBTILES is for a MBTILES container. Defaults to DIRECTORY, unless the output filename has a .mbtiles extension
- **TILE_EXTENSION=value:** Defaults to `pbf`. For tilesets as directories of files, extension of tiles.
- **MINZOOM=<integer>:** Defaults to `0`. Minimum zoom level at which tiles are generated.
- **MAXZOOM=<integer>:** Defaults to `5`. Maximum zoom level at which tiles are generated. Maximum supported value is 22.
- **CONF=[<json>/<filename>]:** Layer configuration as a JSON serialized string. Or, starting with GDAL 3.0.1, filename containing the configuration as JSON .
- **SIMPLIFICATION=float:** Simplification factor for linear or polygonal geometries. The unit is the integer unit of tiles after quantification of geometry coordinates to tile coordinates. Applies to all zoom levels, unless *SIMPLIFICATION_MAX_ZOOM* is also defined.
- **SIMPLIFICATION_MAX_ZOOM=<float>:** Simplification factor for linear or polygonal geometries, that applies only for the maximum zoom level.
- **EXTENT=<positive integer>:** Defaults to `4096`. Number of units in a tile. The greater, the more accurate geometry coordinates (at the expense of tile byte size).
- **BUFFER=<positive integer>:** Number of units for geometry buffering. This value corresponds to a buffer around each side of a tile into which geometries are fetched and clipped. This is used for proper rendering of geometries that spread over tile boundaries by some rendering clients. Defaults to 80 if *EXTENT=4096*.
- **COMPRESS=[YES/NO]:** Defaults to `YES`. Whether to compress tiles with the Deflate/GZip algorithm. Should be left to YES for *FORMAT=MBTILES*.
- **TEMPORARY_DB=<filename>:** Filename with path for the temporary database used for tile generation. By default, this will be a file in the same directory as the output file/directory.
- **MAX_SIZE=<integer>:** Defaults to `500000`. Maximum size of a tile in bytes (after compression). If a tile is greater than this threshold, features will be written with reduced precision, or discarded.
- **MAX_FEATURES=<integer>:** Defaults to `200000`. Maximum number of features per tile.
- **BOUNDS=<min_long,min_lat,max_long,max_lat>:** Override default value for bounds metadata item which is computed from the extent of features written.
- **CENTER=<long,lat,zoom_level>:** Override default value for center metadata item, which is the center of *BOUNDS* at minimum zoom level.
- **TILING_SCHEME=<crs,tile_origin_upper_left_x,tile_origin_upper_left_y,tile_dimension_zoom_0[,tile_matrix_width_zo**
  Define a custom tiling scheme with a CRS (typically given as EPSG:XXXX), the coordinates of the upper-left corner of the upper-left tile (0,0) in the CRS, and the dimension of the tile at zoom level 0. Only available for *FORMAT=DIRECTORY*. The standard WebMercator tiling scheme would be defined by "EPSG:3857,-20037508.343,20037508.343,40075016.686". A tiling scheme for WGS84 geodetic could be "EPSG:4326,-180,180,360". The tiling scheme for Finnish ETRS-TM35FIN (EPSG:3067) is "EPSG:3067,-548576,8388608,2097152". When using such as custom tiling scheme, the 'crs', 'tile_origin_upper_left_x', 'tile_origin_upper_left_y' and 'tile_dimension_zoom_0' entries are added to the metadata.json, and are honoured by the OGR MVT reader. Starting with GDAL 3.10.2, 'tile_matrix_width_zoom_0' (resp. 'tile_matrix_height_zoom_0') can be specified to indicate the number of tiles along the X (resp. Y) axis at zoom level 0.

7.50.7 Layer configuration

The above mentioned CONF dataset creation option can be set to a string whose value is a JSON serialized document such as the below one:

```
{
    "boundaries_lod0": {
        "target_name": "boundaries",
        "description": "Country boundaries",
        "minzoom": 0,
        "maxzoom": 2
    },
    "boundaries_lod1": {
        "target_name": "boundaries",
        "minzoom": 3,
        "maxzoom": 5
    }
}
```

boundaries_lod0 and *boundaries_lod1* are the name of the OGR layers that are created into the target MVT dataset. They are mapped to the MVT target layer *boundaries*.

It is also possible to get the same behavior with the below layer creation options, although that is not convenient in the ogr2ogr use case.

7.50.8 Layer creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALDatasetCreateLayer()`* (C) or *`Dataset.CreateLayer`* (Python). The following layer creation options are supported:

- **MINZOOM=<integer>:** Minimum zoom level at which tiles are generated. Defaults to the dataset creation option *`MINZOOM`* value.
- **MAXZOOM=<integer>:** Maximum zoom level at which tiles are generated. Defaults to the dataset creation option *`MAXZOOM`* value. Maximum supported value is 22.
- **NAME=value:** Target layer name. Defaults to the layer name, but can be overridden so that several OGR layers map to a single target MVT layer. The typical use case is to have different OGR layers for mutually exclusive zoom level ranges.
- **DESCRIPTION=value:** A description of the layer.

7.50.9 Examples

7.50.9.1 Example 1

```
ogrinfo MVT:https://free.tilehosting.com/data/v3/1 -oo tile_extension="pbf.pict?key=$
↪{YOUR_KEY}" --debug on -oo metadata_file="https://free.tilehosting.com/data/v3.json?
↪key=${YOUR_KEY}"
```

7.50.9.2 Example 2

```
ogr2ogr -f MVT mytileset source.gpkg -dsco MAXZOOM=10
```

See Also:

- Mapbox Vector Tile Specification

- *MBTiles* driver
- tippecanoe: Builds vector tilesets from large (or small) collections of GeoJSON, Geobuf, or CSV features
- Links to tools dealing with Mapbox Vector Tiles

7.51 MySQL

Driver short name

MySQL

Build dependencies

MySQL library

This driver implements read and write access for spatial data in MySQL tables.

When opening a database, its name should be specified in the form "MYSQL:dbname[,options]" where the options can include comma separated items like "user=*userid*", "password=*password*", "host=*host*" and "port=*port*".

As well, a "tables=*table*;*table*..." option can be added to restrict access to a specific list of tables in the database. This option is primarily useful when a database has a lot of tables, and scanning all their schemas would take a significant amount of time.

Currently all regular user tables are assumed to be layers from an OGR point of view, with the table names as the layer names. Named views are not currently supported.

If a single integer field is a primary key, it will be used as the FID otherwise the FID will be assigned sequentially, and fetches by FID will be extremely slow.

By default, SQL statements are passed directly to the MySQL database engine. It's also possible to request the driver to handle SQL commands with *OGR SQL* engine, by passing **"OGRSQL"** string to the ExecuteSQL() method, as name of the SQL dialect.

7.51.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

7.51.2 Caveats

- In the case of a layer defined by a SQL statement, fields either named "OGC_FID" or those that are defined as NOT NULL, are a PRIMARY KEY, and are an integer-like field will be assumed to be the FID.
- Geometry fields are read from MySQL using WKB format. Versions older than 5.0.16 of MySQL are known to have issues with some WKB generation and may not work properly.

- The OGR_FID column, which can be overridden with the MYSQL_FID layer creation option, is implemented as a **INT UNIQUE NOT NULL AUTO_INCREMENT** field. This appears to implicitly create an index on the field.
- The geometry column, which defaults to *SHAPE* and can be overridden with the *GEOMETRY_NAME* layer creation option, is created as a **NOT NULL** column in unless *SPATIAL_INDEX* is disabled. By default a spatial index is created at the point the table is created.
- SRS information is stored using the OGC Simple Features for SQL layout, with *geometry_columns* and *spatial_ref_sys* metadata tables being created in the specified database if they do not already exist. The *spatial_ref_sys* table is **not** pre-populated with SRS and EPSG values like PostGIS. If no EPSG code is found for a given table, the MAX(SRID) value will be used. With MySQL 8.0 or later, the *ST_SPATIAL_REFERENCE_SYSTEMS* table provided by the database is used instead of *spatial_ref_sys*.
- Connection timeouts to the server can be specified with the **MYSQL_TIMEOUT** environment variable. For example, SET MYSQL_TIMEOUT=3600. It is possible this variable only has an impact when the OS of the MySQL server is Windows.
- The MySQL driver opens a connection to the database using CLIENT_INTERACTIVE mode. You can adjust this setting (interactive_timeout) in your mysql.ini or mysql.cnf file of your server to your liking.
- We are using WKT to insert geometries into the database. If you are inserting big geometries, you will need to be aware of the *max_allowed_packet* parameter in the MySQL configuration. By default it is set to 1M, but this will not be large enough for really big geometries. If you get an error message like: *Got a packet bigger than 'max_allowed_packet' bytes*, you will need to increase this parameter.

7.51.3 Creation Issues

The MySQL driver does not support creation of new datasets (a database within MySQL), but it does allow creation of new layers within an existing database.

By default, the MySQL driver will attempt to preserve the precision of OGR features when creating and reading MySQL layers. For integer fields with a specified width, it will use **DECIMAL** as the MySQL field type with a specified precision of 0. For real fields, it will use **DOUBLE** with the specified width and precision. For string fields with a specified width, **VARCHAR** will be used.

The MySQL driver makes no allowances for character encodings at this time.

The MySQL driver is not transactional at this time.

7.51.3.1 Layer Creation Options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALDatasetCreateLayer()* (C) or *Dataset.CreateLayer* (Python). The following layer creation options are supported:

- **OVERWRITE=[YES/NO]:** This may be "YES" to force an existing layer of the desired name to be destroyed before creating the requested layer.
- **LAUNDER=[YES/NO]:** Defaults to `YES`. This may be "YES" to force new fields created on this layer to have their field names "laundered" into a form more compatible with MySQL. This converts to lower case and converts some special characters like "-" and "#" to "_". If "NO" exact names are preserved.
- **PRECISION=[TRUE/FALSE]:** Defaults to `TRUE`. This may be "TRUE" to attempt to preserve field widths and precisions for the creation and reading of MySQL layers.
- **GEOMETRY_NAME=value:** Defaults to `SHAPE`. This option specifies the name of the geometry column.
- **FID=value:** Defaults to `OGR_FID`. This option specifies the name of the FID column.

- **FID64=[TRUE/FALSE]:** Defaults to `FALSE`. This may be "TRUE" to create a FID column that can support 64 bit identifiers.
- **SPATIAL_INDEX=[YES/NO]:** May be "NO" to stop automatic creation of a spatial index on the geometry column, allowing NULL geometries and possibly faster loading.
- **ENGINE=value:** Optionally specify database engine to use. In MySQL 4.x this must be set to MyISAM for spatial tables.

Example 1

The following example datasource name opens the database schema *westholland* with password *psv9570* for userid *root* on the port *3306*. No hostname is provided, so localhost is assumed. The tables= directive means that only the bedrijven table is scanned and presented as a layer for use.

```
MYSQL:westholland,user=root,password=psv9570,port=3306,tables=bedrijven
```

Example 2

The following example uses ogr2ogr to create copy the world_borders layer from a shapefile into a MySQL table. It overwrites a table with the existing name *borders2*, sets a layer creation option to specify the geometry column name to *SHAPE2*.

```
ogr2ogr -f MySQL MySQL:test,user=root world_borders.shp -nln borders2 -update -overwrite␣
↪-lco GEOMETRY_NAME=SHAPE2
```

Example 3

The following example uses ogrinfo to return some summary information about the borders2 layer in the test database.

```
$ ogrinfo MySQL:test,user=root borders2 -so

    Layer name: borders2
    Geometry: Polygon
    Feature Count: 3784
    Extent: (-180.000000, -90.000000) - (180.000000, 83.623596)
    Layer SRS WKT:
    GEOGCS["GCS_WGS_1984",
        DATUM["WGS_1984",
            SPHEROID["WGS_84",6378137,298.257223563]],
        PRIMEM["Greenwich",0],
        UNIT["Degree",0.017453292519943295]]
    FID Column = OGR_FID
    Geometry Column = SHAPE2
    cat: Real (0.0)
    fips_cntry: String (80.0)
    cntry_name: String (80.0)
    area: Real (15.2)
    pop_cntry: Real (15.2)
```

7.52 NAS - ALKIS

Driver short name

NAS

Build dependencies

Xerces

The NAS driver reads the NAS/ALKIS format used for cadastral data in Germany. The format is a GML profile with fairly complex GML3 objects not easily read with the general OGR GML driver.

This driver depends on GDAL/OGR being built with the Xerces XML parsing library.

The driver looks for "opengis.net/gml" and one of the strings semicolon separated strings listed in the configuration option **NAS_INDICATOR** (which defaults to "NAS-Operationen;AAA-Fachschema;aaa.xsd;aaa-suite") to determine if a input is a NAS file and ignores all files without any matches.

The configuration option **NAS_GFS_TEMPLATE** makes it possible to cleanly map element paths to feature attributes using a GFS file like in the GML driver. Multiple geometries per layer are also possible (eg. ax_flurstueck.objektkoordinaten next to the regular wkb_geometry). Starting with GDAL 3.7, defining the **NAS_GFS_TEMPLATE** configuration option is required for the NAS driver to open a file. It may be set to the empty string to mean that the driver should try to establish the schema of the file from its content, but using one of templates mentioned below is recommended. Alternatively, starting with GDAL 3.10, specifying the `-if NAS` option to command line utilities accepting it, or `NAS` as the only value of the `papszAllowedDrivers` of *`GDALOpenEx()`*, also forces the driver to recognize the passed filename.

The GFS templates and PostgreSQL schemas are part of norGIS ALKIS-Import (also featuring a shell script and PyQt frontend which ease the import). There are currently two versions:

- GeoInfoDok 6: GFS template (for GDAL >=3.8 and <3.8) and PostgreSQL schema
- GeoInfoDok 7.1.2: GFS template and PostgreSQL schema

The files were generated using xmi2db (fork of xmi2db) from the official application schema.

In GDAL 3.8 the creation of the relation layer *alkis_beziehungen* was removed. Prior the configuration option **NAS_NO_RELATION_LAYER** allowed to disable its population - which was default in ALKIS-Import. The information found there was redundant to the relation fields also contained in original elements/tables. Enabling the option also made progress reporting available.

Duplicate data in datasets will usually causes errors. When importing separate datasets into PostgreSQL it is useful to enable *`OGR_PG_SKIP_CONFLICTS`* to skip conflicting features.

This driver was implemented within the context of the PostNAS project, which has more information on its use and other related projects.

7.52.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.53 NetCDF: Network Common Data Form - Vector

Driver short name

netCDF

Build dependencies

libnetcdf

The netCDF driver supports read and write (creation from scratch and in some cases append operations) to vector datasets (you can find documentation for the *raster side*)

NetCDF is an interface for array-oriented data access and is used for representing scientific data.

The driver handles the "point", "profile" and "trajectory" (single trajectory) feature types of the CF 1.6 convention. For CF-1.7 and below (as well as non-CF files), it also supports a more custom approach for non-point geometries.

The driver also supports writing and reading from CF-1.8 convention compliant files that have simple geometry information encoded within them.

7.53.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.53.2 Conventions and Data Formats

The netCDF vector driver supports reading and writing netCDF files following the Climate and Forecast (CF) Metadata Conventions. Vector datasets can be written using the simple geometry specification of the CF-1.8 convention, or by using the CF-1.6 convention and by writing non-point geometry items as WKT.

7.53.2.1 Distinguishing the Two Formats

Upon reading a netCDF file, the driver will attempt to read the global *Conventions* attribute. If i's value is *CF-1.8* or higher (in this exact format, as specified in the CF convention), then the driver will check if the netCDF file has *CF-1.8* geometries contained within it. If the *Conventions* attribute has a value lower than CF-1.8 or no CF-1.8 geometries, the driver will try to detect "point", "profile" and "trajectory" (single trajectory) feature types.

7.53.2.2 CF-1.8 Writing Limitations

Writing to a CF-1.8 netCDF dataset poses some limitations. Only writing the feature types specified by the CF-1.8 standard (see section *Geometry* for more details) are supported, and measured features are only partially supported. Other geometries, such as non-simple curve geometries, are not supported in any way.

CF-1.8 datasets also do not support the *append* access mode.

There are what are considered *reserved variable names* for CF-1.8 datasets. These variable names are used by the driver to store its metadata. Refrain from using these names as layer names to avoid naming conflicts when writing datasets with multiple layers.

Suppose a layer in a CF-1.8 dataset has the name LAYER with a field with name FIELD. Then the following names would be considered *reserved*:

- *LAYER_node_coordinates*: used to store point information
- *LAYER_node_count*: used to store per shape point count information (not created if LAYER has a geometry type of Point)
- *LAYER_part_node_count*: used to store per part point count information (only created if LAYER consists of MultiLineStrings, MultiPolygons, or has at least one Polygon with interior rings)
- *LAYER_interior_ring*: used to store interior ring information (only created if LAYER consists of at least one Polygon with interior rings)
- *LAYER_field_FIELD*: used to store field information for FIELD.

These names are the only reserved names applying to CF-1.8 datasets.

CF-1.6/WKT datasets are not limited to the aforementioned restrictions.

7.53.3 Mapping of concepts

7.53.3.1 Field types

On creation of netCDF files, the mapping between OGR field types and netCDF type is the following :

OGR field type	netCDF type
String(1)	char
String	char (bi-dimensional), or string for NC4
Integer	int
Integer(Boolean)	byte
Integer(Int16)	short
Integer64	int64 for NC4, or double for NC3 as a fallback
Real	double
Real(Float32)	float
Date	int (with units="days since 1970-1-1")
DateTime	double (with units="seconds since 1970-1-1 0:0:0")

The driver also writes the following attributes for each OGR fields / netCDF variables.

- *ogr_field_name*: OGR field name (useful if the netCDF variable name is different, due to collision)
- *ogr_field_type*: OGR field type (such as String,Integer,Date,DateTime,etc...)
- *ogr_field_width*: OGR field width. Only set if it is non-zero, except for strings
- *ogr_field_precision*: OGR field precision. Only set if it is non-zero
- *standard_name*": (GDAL >= 3.7) OGR field alternative name, if it is compatible with the convention for the value of standard_name.
- *long_name*: (GDAL >= 3.7) OGR field comment, or if not set, OGR field alternative name if it is not compatible with standard_name, or "Field {ogr_field_name}" otherwise.

They are written by default (unless the *WRITE_GDAL_TAGS* dataset creation option is set to NO). They are not required for reading, but may help to better identify field characteristics

On reading, the mapping is the following :

netCDF type	OGR field type
byte	Integer
ubyte (NC4 only)	Integer
char (mono dimensional)	String(1)
char (bi dimensional)	String
string (NC4 only)	String
short	Integer(Int16)
ushort (NC4 only)	Integer
int	Integer
int or double (with units="days since 1970-1-1")	Date
uint (NC4 only)	Integer64
int64 (NC4 only)	Integer64
uint64 (NC4 only)	Real
float	Real(Float32)
double	Real
double (with units="seconds since 1970-1-1 0:0:0")	DateTime

7.53.3.2 Layers

In the CF-1.8 compliant driver, a single layer corresponds to a single **geometry container** within a CF-1.8 compliant netCDF file. A geometry container, per the CF-1.8 specification, is referred to by another variable (presumably a data variable) through the **geometry** attribute. When reading a CF-1.8 compliant netCDF file, all geometry containers within the netCDF file will be present in the opened dataset as separate layers. Similarly, when writing to a CF-1.8 dataset, each layer will be written to a geometry container whose variable name is that of the source layer. When writing to a CF-1.8 dataset specifically, multiple layers are always enabled and are always in a single netCDF file, regardless of the `MULTIPLE_LAYERS` option.

When working with files made with older versions of the driver (pre CF-1.8), a single netCDF file generally corresponds to a single OGR layer, provided that it contains only mono-dimensional variables, indexed by the same dimension (or bi-dimensional variables of type char). For netCDF v4 files with multiple groups, each group may be seen as a separate OGR layer. On writing, the `MULTIPLE_LAYERS` dataset creation option can be used to control whether multiple layers is disabled, or if multiple layers should go in separate files, or separate groups.

7.53.3.3 Strings

Variable length strings are not natively supported in netCDF v3 format. To work around that, OGR uses bi-dimensional char variables, whose first dimension is the record dimension, and second dimension the maximum width of the string.

By default, OGR implements a "auto-grow" mode in writing, where the maximum width of the variable used to store a OGR string field is extended when needed.

For WKT datasets, this leads to a full rewrite of already written records; although this process is transparent for the user, it can slow down the creation process in non-linear ways. A similar mechanism is used to handle layers with geometry types other than point to store the ISO WKT representation of the geometries.

For CF-1.8 datasets, growing the string width dimension is a relatively inexpensive process which does not involve recopying of records, but involves only a simple integer reassignment. Because of how inexpensive dimension growth is with CF-1.8 datasets, auto growth of the string width dimension is always on.

When using a netCDF v4 output format (NC4), strings will be by default written as netCDF v4 variable length strings.

7.53.3.4 Geometry

Supported feature types when reading from a CF-1.8 convention compliant netCDF file include OGRPoint, OGRLineString, OGRPolygon, OGRMultiPoint, OGRMultiLineString, and OGRMultiPolygon. Due to slight ambiguities present in the CF-1.8 convention concerning Polygons versus MultiPolygons, the driver will in most cases default to assuming a MultiPolygon for the geometry of a layer with **geometry_type** polygon. The one exception where a Polygon type will be used is when the attribute **part_node_count** is not present within that layer's geometry container. Per convention requirements, the driver supports reading and writing from geometries with X, Y, and Z axes. Writing from source layers with features containing an M axis is also partially supported. The X, Y, and Z information of a measured feature will be able to be captured in a CF-1.8 netCDF file, but the measure information will be lost completely.

When working with a CF-1.6/WKT dataset, layers with a geometry type of Point or Point25D will cause the implicit creation of x,y(,z) variables for a projected coordinate system, or lon,lat(,z) variables for geographic coordinate systems. For other geometry types, a variable "ogc_wkt" (bi-dimensional char for NC3 output, or string for NC4 output) is created and used to store the geometry as a ISO WKT string.

7.53.3.5 "Profile" feature type

The driver can handle "profile" feature type, i.e. phenomenons that happen at a few positions along a vertical line at a fixed horizontal position. In that representation, some variables are indexed by the profile, and others by the observation.

More precisely, the driver supports reading and writing profiles organized accordingly with the "Indexed ragged array representation" of profiles.

On reading, the driver will collect values of variables indexed by the profile dimension and expose them as long as variables indexed by the observation dimension, based on a variable such as "parentIndex" with an attribute "instance_dimension" pointing to the profile dimension.

On writing, the *FEATURE_TYPE*=PROFILE layer creation option must be set and the driver will need to be instructed which OGR fields are indexed either by the profile or by the observation dimension. The list of fields indexed by the profile can be specified with the *PROFILE_VARIABLES* layer creation options (other fields are assumed to be indexed by the observation dimension). Fields indexed by the profile are the horizontal geolocation (created implicitly), and other user attributes such as the location name, etc. Care should be taken into selecting which variables are indexed by the profile dimension: given 2 OGR features (taking into account only the variables indexed by the profile dimension), if they have different values for such variables, they will be considered to belong to different profiles.

In the below example, the station_name and time variables may be indexed by the profile dimension (the geometry is assumed to be also indexed by the profile dimension), since all records that have the same value for one of those variables have same values for the other ones, whereas temperature and Z should be indexed by the default dimension.

station_name	time	geometry	temperature	Z
Paris	2016-03-01T00:00:00Z	POINT (2 49)	25	100
Vancouver	2016-04-01T12:00:00Z	POINT (-123 49.25)	5	100
Paris	2016-03-01T00:00:00Z	POINT (2 49)	3	500
Vancouver	2016-04-01T12:00:00Z	POINT (-123 49.25)	-15	500

An integer field, with the name of the profile dimension (whose default name is "profile", which can be altered with the *PROFILE_DIM_NAME* layer creation option), will be used to store the automatically computed id of profile sites (unless a integer OGR field with the same name exits).

The size of the profile dimension defaults to 100 for non-NC4 output format, and is extended automatically in case of additional profiles (with similar performance issues as growing strings). For NC4 output format, the profile dimension is of unlimited size by default.

7.53.4 Dataset creation options

Dataset creation options can be specified in command-line tools using the syntax `-dsco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following dataset creation options are supported:

- **GEOMETRY_ENCODING=[CF_1.8/WKT]:** Defaults to `CF_1.8`. Chooses which geometry encoding to use when creating new layers within the dataset.
- **FORMAT=[NC/NC2/NC4/NC4C]:** Defaults to `Nc`. netCDF format. NC is the classic netCDF format (compatible with netCDF v3.X and 4.X libraries). NC2 is the extension of NC for files larger than 4 GB. NC4 is the netCDF v4 format, using a HDF5 container, offering new capabilities (new types, concept of groups, etc...) only available in netCDF v4 library. NC4C is a restriction of the NC4 format to the concepts supported by the classic netCDF format.
- **WRITE_GDAL_TAGS=[YES/NO]:** Defaults to `YES`. Whether to write GDAL specific information as netCDF attributes.
- **CONFIG_FILE=[<filename>/<xml>]:** Path to a *XML configuration file* (or its content inlined) for precise control of the output.

The following option will only have effect when simultaneously specifying *`GEOMETRY_ENCODING=WKT`*:

- **MULTIPLE_LAYERS=[NO/SEPARATE_FILES/SEPARATE_GROUPS]:** Defaults to `NO`. Default is NO, i.e a dataset can contain only a single OGR layer. SEPARATE_FILES can be used to put the content of each OGR layer in a single netCDF file, in which case the name passed at dataset creation is used as the directory, and the layer name is used as the basename of the netCDF file. SEPARATE_GROUPS may be used when *`FORMAT=NC4`* to put each OGR layer in a separate netCDF group, inside the same file.

7.53.5 Layer creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALDatasetCreateLayer()`* (C) or *`Dataset.CreateLayer`* (Python).

The following option applies to both dataset types:

- **USE_STRING_IN_NC4=[YES/NO]:** Whether to use NetCDF string type for strings in NC4 format. If NO, bidimensional char variable are used. Default to YES when *`FORMAT=NC4`*.

The following options require a dataset with *`GEOMETRY_ENCODING=WKT`*:

- **RECORD_DIM_NAME=value:** Defaults to `record`. Name of the unlimited dimension that index features.

- **STRING_DEFAULT_WIDTH=<integer>:** Default width of strings (when using bi-dimensional char variables). Default is 10 in autogrow mode, 80 otherwise.
- **WKT_DEFAULT_WIDTH=<integer>:** Default width of WKT strings (when using bi-dimensional char variables). Default is 1000 in autogrow mode, 10000 otherwise.
- **AUTOGROW_STRINGS=[YES/NO]:** Defaults to `YES`. Whether to auto-grow string fields of non-fixed width, or ogc_wkt special field, when serialized as bidimensional char variables. When set to NO, if the string is larger than its maximum initial width (set by STRING_DEFAULT_WIDTH), it is truncated. For a geometry, it is completely discarded.
- **FEATURE_TYPE=[AUTO/POINT/PROFILE]:** Defaults to `AUTO`. Select the CF FeatureType. Defaults to AUTO where FeatureType=Point is selected if the layer geometry type is Point, otherwise the custom approach involving the "ogc_wkt" field is used. Can be set to *PROFILE* so as to select the creation of an indexed ragged array representation of profiles.
- **PROFILE_DIM_NAME=value:** Defaults to `profile`. Name of the profile dimension and variable. Only used when *`FEATURE_TYPE=PROFILE`*.
- **PROFILE_DIM_INIT_SIZE=value:** Initial size of profile dimension, or UNLIMITED for NC4 files. Defaults to 100 when FORMAT != NC4 and to UNLIMITED when FORMAT = NC4. Only used when *`FEATURE_TYPE=PROFILE`*.
- **PROFILE_VARIABLES=value:** Comma separated list of field names that must be indexed by the profile dimension. Only used when *`FEATURE_TYPE=PROFILE`*.

The following option requires a dataset with GEOMETRY_ENCODING=CF_1.8:

- **BUFFER_SIZE=<bytes>:** The soft limit of the write buffer in bytes. Larger values generally imply better performance, but values should be comfortably less than that of available physical memory or else thrashing can occur. By default, this value is set at 20% of usable physical memory (usable meaning total physical RAM considering limitations of virtual address space size). Buffer contents are committed between translating features, but not *during* translating a feature, so this limit does not apply to a single feature. The minimum acceptable size is 4096. If a value lower than this is specified the default will be used.
- **GROUPLESS_WRITE_BACK=[YES/NO]:** Defaults to `NO`. In order to reduce time used to write data to the target netCDF file, data is often grouped together in arrays and written all at once. Each of these arrays is associated with a variable in the target dataset. Arrays are destroyed as soon as the associated data is written to the netCDF file which in turn occurs as soon as a complete data array for a variable is assembled in memory. For machines with small memory sizes, this optimization may cause issues when writing large datasets with large layers. Turning this option on by specifying "YES" disables array writing and causes data to be written one datum at a time. It is strongly recommended to keep this option off unless out of memory errors or performance issues occur. In the general case, this technique greatly improves translation efficiency.

7.53.6 XML configuration file

A XML configuration file conforming to the schema specified in netcdf_config.xsd can be used for very precise control on the output format, in particular to set all needed attributes (such as units) to conform to the NetCDF CF-1.6 convention.

It has been designed in particular, but not exclusively, to be usable in use cases involving the MapServer OGR output.

Such a file can be used to :

- set dataset and layer creation options.
- set global netCDF attributes.
- map OGR field names to netCDF variable names.
- set netCDF attributes attached to netCDF variables.

The scope of effect is either globally, when elements are defined as direct children of the root <Configuration> node, or specifically to a given layer, when defined as children of a <Layer> node.

The filename is specified with the CONFIG_FILE dataset creation option. Alternatively, the content of the file can be specified inline as the value of the option (it must then begin strictly with the "<Configuration" characters)

The following example shows all possibilities and precedence rules:

```
<Configuration>
    <DatasetCreationOption name="FORMAT" value="NC4"/>
    <DatasetCreationOption name="MULTIPLE_LAYERS" value="SEPARATE_GROUPS"/>
    <LayerCreationOption name="RECORD_DIM_NAME" value="observation"/>
<!-- applies to all layers -->
    <Attribute name="copyright" value="Copyright(C) 2016 Example"/>
    <Field name="weight">  <!-- edit user field/variable -->
        <Attribute name="units" value="kg"/>
        <Attribute name="maximum" value="10" type="double"/>
    </Field>
    <Field netcdf_name="z"> <!-- edit predefined variable -->
        <Attribute name="long_name" value="Elevation"/>
    </Field>
<!-- start of layer specific definitions -->
    <Layer name="1st_layer" netcdf_name="firstlayer"> <!-- OGR layer "1st_layer" is␣
↪renamed as "firstlayer" netCDF group -->
        <LayerCreationOption name="FEATURE_TYPE" value="POINT"/>
        <Attribute name="copyright" value="Public domain"/> <!-- override global one -->
        <Attribute name="description" value="This is my first layer"/> <!-- additional␣
↪attribute -->
        <Field name="1st_field" netcdf_name="firstfield"/> <!-- rename OGR field "1st_
↪field" as the "firstfield" netCDF variable -->
        <Field name="weight"/> <!-- cancel above global customization -->
        <Field netcdf_name="lat"> <!-- edit predefined variable -->
            <Attribute name="long_name" value=""/> <!-- remove predefined attribute -->
        </Field>
    </Layer>
    <Layer name="sounding">
        <LayerCreationOption name="FEATURE_TYPE" value="PROFILE"/>
        <Field name="station_name" main_dim="profile"/> <!-- the corresponding netCDF␣
↪variable will be indexed against the profile dimension, instead of the observation␣
↪dimension -->
        <Field name="time" main_dim="profile"/> <!-- the corresponding netCDF variable␣
↪will be indexed against the profile dimension, instead of the observation dimension -->
    </Layer>
</Configuration>
```

The effect on the output can be checked by running the **ncdump** utility

7.53.7 Further Reading

- *Raster side of the netCDF driver.*
- NetCDF CF-1.6 convention
- NetCDF CF-1.8 convention draft
- NetCDF compiled libraries

- NetCDF Documentation

7.53.8 Credits

Development of the read/write vector capabilities for netCDF was funded by Meteorological Service of Canada , World Ozone and Ultraviolet Radiation Data Centre, and the US Geological Survey.

7.54 NGW -- NextGIS Web

Driver short name

NGW

Build dependencies

libcurl

NextGIS Web - is a server GIS, which allows storing and editing geodata and to display maps in web browser. Also NextGIS Web can share geodata with other NextGIS software.

NextGIS Web has the following features:

- Display maps in a web browser (different maps with different layers and styles)
- Flexible permissions management
- Load geodata from PostGIS or import from GIS formats (ESRI Shape, GeoJSON or GeoTIFF)
- Load vector geodata in the following formats: GeoJSON, CSV, ESRI Shape, Mapinfo tab
- Import map styles from QGIS project or set them manually
- Act as a server for TMS, WMS, MVT, WFS
- Act as a client for WMS
- User can add photos to records, change record attributes via web interface or WFS-T protocol

NextGIS Web - is an open source software (license GPL v2+, see GNU General Public License, version 2).

7.54.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

7.54.2 Driver

The driver can connect to the services implementing the NextGIS Web REST API. NGW driver requires cURL support in GDAL and WMS driver. The driver supports read and write operations.

7.54.3 Dataset name syntax

The minimal syntax to open a NGW datasource is: NGW:[NextGIS Web URL][/resource/][resource identifier]

- **NextGIS Web URL** may be an URL to nextgis.com cloud service (for example, https://demo.nextgis.com), or some other URL including port and additional path (for example, http://192.168.1.1:8000/test).
- **resource** is mandatory keyword dividing resource identifier from the rest of URL.
- **resource identifier** this is positive number from 0 and above. This may be a resource group, vector, PostGIS or raster layer, style.

All vector layers, PostGIS, raster layers, styles will list as child resources if identifier is resource group. In other case this will be a separate layer.

7.54.4 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The following configuration options are available:

- **NGW_USERPWD=value:** User name and password separated with colon. Optional and can be set using open options.
- **NGW_BATCH_SIZE=value:** Size of feature insert and update operations cache before send to server. If batch size is -1 batch mode is disabled. Delete operation will execute immediately.
- **NGW_PAGE_SIZE=value:** If supported by server, fetch features from remote server will use paging. The -1 value disables paging even it supported by server.
- **NGW_NATIVE_DATA=value:** Whether to store the json *extensions* key in feature native data.
- **NGW_JSON_DEPTH=value:** The depth of json response that can be parsed. If depth is greater than this value, parse error occurs.

  The depth of json response that can be parsed. If depth is greater than this value, parse error occurs.
- **NGW_EXTENSIONS=value:** Comma separated extensions list. Available values are *description* and *attachment*. This needed to fill native data.
- **NGW_CONNECTTIMEOUT=value:** Maximum delay for the connection to be established before being aborted in seconds.
- **NGW_TIMEOUT=value:** Maximum delay for the whole request to complete before being aborted in seconds.
- **NGW_MAX_RETRY=value:** Maximum number of retry attempts if a 429, 502, 503 or 504 HTTP error occurs.
- **NGW_RETRY_DELAY=value:** Number of seconds between retry attempts.

7.54.5 Authentication

Any operations (read, write, get metadata, change properties, etc.) may require an authenticated access. Authenticated access is obtained by specifying user name and password in open, create or configuration options.

7.54.6 Feature

If the `NATIVE_DATA` open option is set to YES, the *extensions* json object will store as a serialized json object in the NativeData property of the OGRFeature object (and "application/json" in the NativeMediaType property). If writing OGRFeature has NativeMediaType property set to "application/json" and its NativeData property set to serialized json object the new NGW feature *extensions* json object will fill from this json object.

Extensions json object structure see in NextGIS Web API documentation

7.54.7 Geometry

NextGIS Web supports only one geometry column. Default spatial reference is Web Mercator (EPSG:3857). The following geometry types are available:

- POINT
- LINESTRING
- POLYGON
- MULTIPOINT
- MULTILINESTRING
- MULTIPOLYGON

Geometry with Z value also supported.

7.54.8 Field data types

NextGIS Web supports only following field types:

- OFTInteger
- OFTInteger64
- OFTReal
- OFTString
- OFTDate
- OFTTime
- OFTDateTime

Driver stores additional field data in comment as JSON string:

- field identifier in NGW
- check if this is label field
- check to show field in grid view
- check to use field in text search

Driver supports alter field:

- name
- alternative name

- field domain
- comment

7.54.9 Paging

Features can retrieved from NextGIS Web by chunks if supported by server (available since NextGIS Web 3.1). The chunk size can be altered with the *NGW_PAGE_SIZE* configuration option or *PAGE_SIZE* open option.

7.54.10 Write support

Datasource and layers creation and deletion is possible. Write support is only enabled when the datasource is opened in update mode and user has appropriate permissions. Vector and PostGIS layers insert and update operations are cached if *BATCH_SIZE* is greater 0. Delete operation executes immediately.

7.54.11 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are available:

- **USERPWD=value:** Username and password, separated by colon.
- **PAGE_SIZE=value:** Defaults to `-1`. Limit feature count while fetching from server. Default value is -1 - no limit.
- **BATCH_SIZE=value:** Defaults to `-1`. Size of feature insert and update operations cache before send to server. If batch size is -1 batch mode is disabled.
- **NATIVE_DATA=[YES/NO]:** Defaults to `NO`. Whether to store the json *extensions* key in feature native data.
- **JSON_DEPTH=value:** Defaults to `32`. The depth of json response that can be parsed. If depth is greater than this value, parse error occurs.
- **EXTENSIONS=value:** Comma separated extensions list. Available values are *description* and *attachment*. This needed to fill native data.
- **CONNECTTIMEOUT=value:** Maximum delay for the connection to be established before being aborted in seconds.
- **TIMEOUT=value:** Maximum delay for the whole request to complete before being aborted in seconds.
- **MAX_RETRY=value:** Maximum number of retry attempts if a 429, 502, 503 or 504 HTTP error occurs.
- **RETRY_DELAY=value:** Number of seconds between retry attempts.

7.54.12 Dataset creation options

Dataset creation options can be specified in command-line tools using the syntax `-dsco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following dataset creation options are available:

- **KEY=value:** Key value. Must be unique in whole NextGIS Web instance. Optional.
- **DESCRIPTION=value:** Resource description. Optional.
- **USERPWD=value:** Username and password, separated by colon.
- **PAGE_SIZE=value:** Defaults to `-1`. Limit feature count while fetching from server. Default value is -1 - no limit.
- **BATCH_SIZE=value:** Defaults to `-1`. Size of feature insert and update operations cache before send to server. If batch size is -1 batch mode is disable.

- **NATIVE_DATA=[YES/NO]:** Defaults to `NO`. Whether to store the json *extensions* key in feature native data.
- **JSON_DEPTH=value:** Defaults to `32`. The depth of json response that can be parsed. If depth is greater than this value, parse error occurs.
- **EXTENSIONS=value:** Comma separated extensions list. Available values are *description* and *attachment*. This needed to fill native data.
- **CONNECTTIMEOUT=value:** Maximum delay for the connection to be established before being aborted in seconds.
- **TIMEOUT=value:** Maximum delay for the whole request to complete before being aborted in seconds.
- **MAX_RETRY=value:** Maximum number of retry attempts if a 429, 502, 503 or 504 HTTP error occurs.
- **RETRY_DELAY=value:** Number of seconds between retry attempts.

7.54.13 Layer creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALDatasetCreateLayer()`* (C) or *`Dataset.CreateLayer`* (Python). The following layer creation options are available:

- **OVERWRITE=[YES/NO]:** Defaults to `NO`. Whether to overwrite an existing table with the layer name to be created. The resource will delete and new one will created. This leads that resource identifier will change. Defaults to
- **KEY=value:** Key value. Must be unique in whole NextGIS Web instance. Optional.
- **DESCRIPTION=value:** Resource description. Optional.

7.54.14 Metadata

NextGIS Web metadata are supported in datasource, vector, PostGIS, raster layers and styles. Metadata are stored at specific domain "NGW". NextGIS Web supported metadata are strings and numbers. Metadata keys with decimal numbers will have suffix **.d** and for real numbers - **.f**. To create new metadata item, add new key=value pair in NGW domain using the *SetMetadataItem* function and appropriate suffix. During transferring to NextGIS Web, suffix will be omitted. You must ensure that numbers correctly transform from string to number.

Resource description and key map to appropriate *description* and *keyname* metadata items in default domain. Changing those metadata items will cause an update of resource properties.

Resource creation date, type and parent identifier map to appropriate read-only metadata items *creation_date*, *resource_type* and *parent_id* in default domain.

Vector layer field properties (identifier, label field, grid visibility, text search) saved as json string in field comment.

Starting from GDAL 3.3 field alias can be set/get via *SetAlternativeName* and *GetAlternativeNameRef*.

7.54.15 Domains

Driver supports only coded field domain. Since NGW does not support field types in domains, three domains are created for each domain where keys can be represented as numbers:

- domain_name with field type OFTString
- domain_name + " (number)" with field type OFTInteger
- domain_name + " (bigint)" with field type OFTString64

Deleting any of the three domains will delete the others.

Also NGW does not support null as coded values. So the null will represent as empty string.

7.54.16 Filters

Vector and PostGIS layers support SetIgnoredFields method. When this method executes any cached features will be freed.

Vector and PostGIS layers support SetAttributeFilter and SetSpatialFilter methods. The attribute filter will evaluate at server side if condition is one of following comparison operators:

- greater (>)
- lower (<)
- greater or equal (>=)
- lower or equal (<=)
- equal (=)
- not equal (!=)
- LIKE SQL statement (for strings compare)
- ILIKE SQL statement (for strings compare)

Also only AND operator without brackets supported between comparison. For example,

```
FIELD_1 = 'Value 1'
```

```
FIELD_1 = 'Value 1' AND FIELD_2 > Value 2
```

In other cases attribute filter will evaluate on client side.

You can set attribute filter using NextGIS Web native format. For example,

```
NGW:fld_FIELD_1=Value 1&fld_FIELD_2__gt=Value 2
```

Don't forget to add 'NGW:' perefix to where clause and 'fld_' prefix to field name.

Dataset supports ExecuteSQL method. Only the following queries are supported:

- DELLAYER: layer_name; - delete layer with layer_name.
- DELETE FROM layer_name; - delete any features from layer with layer_name.
- DELETE FROM layer_name WHERE field = value; - delete features from layer with layer_name and where clause.
- DROP TABLE layer_name; - delete layer with layer_name.
- ALTER TABLE src_layer RENAME TO dst_layer; - rename layer.
- SELECT field_1,field_2 FROM src_layer WHERE field_1 = 'Value 1' AND field_2 = 'Value 2';

In SELECT statement field list or asterisk can be provided. The WHERE clause has same limitations as SetAttributeFilter method input.

7.54.17 Examples

7.54.17.1 Example 1: Reading datasource contents

(1730 is resource group identifier):

```
ogrinfo -ro NGW:https://demo.nextgis.com/resource/1730
```

7.54.17.2 Example 2: Reading layer details

(*1730* is resource group identifier, *Parks* is vector layer name):

```
ogrinfo -ro -so NGW:https://demo.nextgis.com/resource/1730 Parks
```

7.54.17.3 Example 3: Creating and populating a vector layer from a shapefile in existing resource group with identifier 1730.

New vector layer name will be "some new name":

```
ogr2ogr -f NGW -nln "some new name" -update -doo "BATCH_SIZE=100" -t_srs EPSG:3857
↪"NGW:https://demo.nextgis.com/resource/1730" myshapefile.shp
```

Warning

The *-update* key is mandatory, otherwise the destination datasource will silently delete. The *-t_srs EPSG:3857* key is mandatory because vector layers spatial reference in NextGIS Web can be only in EPSG:3857.

Note

The *-doo "BATCH_SIZE=100"* key is recommended for speed up feature transferring.

7.54.17.4 Example 4

Creating and populating a vector layer from a shapefile in new resource group with name "new group" and parent identifier 1730.

New vector layer name will be "some new name".

```
ogr2ogr -f NGW -nln "Название на русском языке" -dsco "BATCH_SIZE=100" -t_srs EPSG:3857
↪"NGW:https://demo.nextgis.com/resource/1730/new group" myshapefile.shp
```

7.54.18 See also

- *Raster side of the driver*
- NextGIS Web documentation
- NextGIS Web for developers

7.55 UK .NTF

Driver short name

UK .NTF

Driver built-in by default

This driver is built-in by default

The National Transfer Format, mostly used by the UK Ordnance Survey, is supported for read access.

This driver treats a directory as a dataset and attempts to merge all the .NTF files in the directory, producing a layer for each type of feature (but generally not for each source file). Thus a directory containing several Landline files will have three layers (LANDLINE_POINT, LANDLINE_LINE and LANDLINE_NAME) regardless of the number of landline files.

NTF features are always returned with the British National Grid coordinate system. This may be inappropriate for NTF files written by organizations other than the UK Ordnance Survey.

7.55.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.55.2 See Also

- General UK NTF Information

7.55.3 Implementation Notes

7.55.3.1 Products (and Layers) Supported

```
Landline (and Landline Plus):
    LANDLINE_POINT
    LANDLINE_LINE
    LANDLINE_NAME

Panorama Contours:
    PANORAMA_POINT
    PANORAMA_CONTOUR

    HEIGHT attribute holds elevation.

Strategi:
    STRATEGI_POINT
    STRATEGI_LINE
    STRATEGI_TEXT
    STRATEGI_NODE

Meridian:
    MERIDIAN_POINT
    MERIDIAN_LINE
    MERIDIAN_TEXT
    MERIDIAN_NODE

Boundaryline:
```

(continues on next page)

(continued from previous page)

```
    BOUNDARYLINE_LINK
    BOUNDARYLINE_POLY
    BOUNDARYLINE_COLLECTIONS

    The _POLY layer has links to links allowing true polygons to
        be formed (otherwise the _POLY's only have a seed point for geometry.
    The collections are collections of polygons (also without geometry
    as read).  This is the only product from which polygons can be
    constructed.

BaseData.GB:
    BASEDATA_POINT
    BASEDATA_LINE
    BASEDATA_TEXT
    BASEDATA_NODE

OSCAR Asset/Traffic:
    OSCAR_POINT
    OSCAR_LINE
    OSCAR_NODE

OSCAR Network:
    OSCAR_NETWORK_POINT
    OSCAR_NETWORK_LINE
    OSCAR_NETWORK_NODE

Address Point:
    ADDRESS_POINT

Code Point:
    CODE_POINT

Code Point Plus:
    CODE_POINT_PLUS
```

The dataset as a whole will also have a FEATURE_CLASSES layer containing a pure table relating FEAT_CODE numbers with feature class names (FC_NAME). This applies to all products in the dataset. A few layer types (such as the Code Point, and Address Point products) don't include feature classes. Some products use features classes that are not defined in the file, and so they will not appear in the FEATURE_CLASSES layer.

7.55.3.2 Product Schemas

The approach taken in this reader is to treat one file, or a directory of files as a single dataset. All files in the dataset are scanned on open. For each particular product (listed above) a set of layers are created; however, these layers may be extracted from several files of the same product.

The layers are based on a low level feature type in the NTF file, but will generally contain features of many different feature codes (FEAT_CODE attribute). Different features within a given layer may have a variety of attributes in the file; however, the schema is established based on the union of all attributes possible within features of a particular type (i.e. POINT) of that product family (i.e. OSCAR Network).

If an NTF product is read that doesn't match one of the known schema's it will go through a different generic handler which has only layers of type GENERIC_POINT and GENERIC_LINE. The features only have a FEAT_CODE attribute.

Details of what layers of what products have what attributes can be found in the NTFFileReader::EstablishLayers() method at the end of ntf_estlayers.cpp. This file also contains all the product specific translation code.

7.55.3.3 Special Attributes

```
FEAT_CODE: General feature code integer, can be used to lookup a name in the
           FEATURE_CLASSES layer/table.

TEXT_ID/POINT_ID/LINE_ID/NAME_ID/COLL_ID/POLY_ID/GEOM_ID:
          Unique identifier for a feature of the appropriate type.

TILE_REF: All layers (except FEATURE_CLASSES) contain a TILE_REF attribute
          which indicates which tile (file) the features came from.  Generally
          speaking the id numbers are only unique within the tile and so
          the TILE_REF can be used restrict id links within features from
          the same file.

FONT/TEXT_HT/DIG_POSTN/ORIENT:
    Detailed information on the font, text height, digitizing position,
        and orientation of text or name objects.  Review the OS product
        manuals to understand the units, and meaning of these codes.

GEOM_ID_OF_POINT:
    For _NODE features this defines the POINT_ID of the point layer object
        to which this node corresponds.  Generally speaking the nodes don't
        carry a geometry of their own.  The node must be related to a point
        to establish its position.

GEOM_ID_OF_LINK:
    A _list_ of _LINK or _LINE features to end/start at a node.  Nodes,
        and this field are generally only of value when establishing
        connectivity of line features for network analysis.   Note that this
        should be related to the target features GEOM_ID, not its LINE_ID.

        On the BOUNDARYLINE_POLY layer this attribute contains the GEOM_IDs
        of the lines which form the edge of the polygon.

POLY_ID:
    A list of POLY_ID's from the BOUNDARYLINE_POLY layer associated with
        a given collection in the BOUNDARYLINE_COLLECTIONS layer.
```

7.55.3.4 Generic Products

In situations where a file is not identified as being part of an existing known product it will be treated generically. In this case the entire dataset is scanned to establish what features have what attributes. Because of this, opening a generic dataset can be much slower than opening a recognised dataset. Based on this scan a list of generic features (layers) are defined from the following set:

```
GENERIC_POINT
GENERIC_LINE
GENERIC_NAME
GENERIC_TEXT
GENERIC_POLY
```

(continues on next page)

(continued from previous page)

```
GENERIC_NODE
GENERIC_COLLECTION
```

Generic products are primarily handled by the ntf_generic.cpp module whereas specific products are handled in ntf_estlayers.cpp.

Because some data products (OSNI datasets) not from the Ordnance Survey were found to have record groups in unusual orders compared to what the UK Ordnance Survey does, it was found necessary to cache all the records of level 3 and higher generic products, and construct record groups by id reference from within this cache rather than depending on convenient record orderings. This is accomplished by the NTFFileReader "indexing" capability near the bottom of ntffilereader.cpp. Because of this in memory indexing accessing generic datasets can be much more memory intensive than accessing known data products, though it isn't necessary for generic level 1 and 2 products.

It is possible to force a known product to be treated as generic by setting the FORCE_GENERIC option to "ON" using OGRNTFDataSource::SetOptionsList() as is demonstrated in ntfdump.cpp. This may also be accomplished from outside OGR applications by setting the OGR_NTF_OPTIONS environment variable to "FORCE_GENERIC=ON".

7.56 OGC API - Features

Driver short name

OAPIF

Build dependencies

libcurl

This driver can connect to a OGC API - Features service. It assumes that the service supports OpenAPI 3.0/JSON/GeoJSON encoding for respectively API description, feature collection metadata and feature collection data.

Note

In versions prior to GDAL 3.1, this driver was called the WFS3 driver, and only supported draft versions of the specification.

7.56.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

7.56.2 Dataset name syntax

The syntax to open a OGC API - Features datasource is : *OAPIF:http://path/to/OAPIF/endpoint*

where endpoint is the landing page or a the path to collections/{id}.

Starting with GDAL 3.10, specifying the `-if OAPIF` option to command line utilities accepting it, or `OAPIF` as the only value of the `papszAllowedDrivers` of *`GDALOpenEx()`*, also forces the driver to recognize the passed URL without

the `OAPIF:` prefix.

7.56.3 Layer schema

OGR needs a fixed schema per layer, but OGC API - Features Core doesn't impose fixed schema. The driver will use the XML schema or JSON schema pointed by the "describedby" relationship of a collection, if it exists. The driver will also retrieve the first page of features (using the selected page) and establish a schema from this.

7.56.4 Filtering

The driver will forward any spatial filter set with SetSpatialFilter() to the server. In OGC API - Features Core, only a subset of attributes allowed by the server can be queried for equalities, potentially combined with a AND logical operator. More complex requests will be partly or completely evaluated on client-side.

Rectangular spatial filtering is forward to the server as well.

Starting with GDAL 3.10, the `DATETIME` open option can also be specified for temporal filtering.

7.56.5 CRS support

Starting with GDAL 3.7, the driver supports the OGC API - Features - Part 2: Coordinate Reference Systems by Reference extension. If a server reports a storageCRS property, that property will be used to set the CRS of the OGR layer. Otherwise the default will be OGC:CRS84 (WGS84 longitude, latitude). As most all OGR drivers, the OAPIF driver will report the SRS and geometries, and expect spatial filters, in the "GIS-friendly" order, with longitude/easting first (X component), latitude/northing second (Y component), potentially overriding the axis order of the authority.

The CRS of layers can also be controlled with the *`CRS`* or *`PREFERRED_CRS`* open options documented below.

7.56.6 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are available:

- **URL=value:** URL to the OGC API - Features server landing page, or to a given collection. Required when using the "OAPIF:" string as the connection string.
- **PAGE_SIZE=<integer>:** Defaults to `1000`. Number of features to retrieve per request. Minimum is 1. If not set, an attempt to determine the maximum allowed size will be done by examining the API schema.
- **INITIAL_REQUEST_PAGE_SIZE=<integer>:** Defaults to `20`. Number of features to retrieve during the initial request done in order to retrieve information about the features. Minimum is 1. Maximum is the value of the *`PAGE_SIZE`* option. If not set the default (20) will be used.
- **USERPWD=value:** May be supplied with *userid:password* to pass a userid and password to the remote server.
- **IGNORE_SCHEMA=[YES/NO]:** (GDAL >= 3.1) Set to YES to ignore the XML Schema or JSON schema that may be offered by the server.
- **CRS=value:** (GDAL >= 3.7) Set to a CRS identifier, e.g `EPSG:3067` or `http://www.opengis.net/def/crs/EPSG/0/3067`, to use as the layer CRS. That CRS must be listed in the lists of CRS supported by the layers of the dataset, otherwise layers not listing it cannot be opened.
- **PREFERRED_CRS=value:** (GDAL >= 3.7) Identical to the *`CRS`* option, except that if a layer does not list the PREFERRED_CRS in its list of supported CRS, the default CRS (storageCRS when present, otherwise EPSG:4326) will be used. *`CRS`* and *`PREFERRED_CRS`* option are mutually exclusive.
- **SERVER_FEATURE_AXIS_ORDER=[AUTHORITY_COMPLIANT/GIS_FRIENDLY]:** Defaults to `AUTHORITY_COMPLIANT`. This option can be set to GIS_FRIENDLY if axis order issue are noticed in features received from the server, indicating that the server does not return them in the axis order mandated by the CRS

authority, but in a more traditional "GIS friendly" order, with longitude/easting first, latitude/northing second. Do not set this option unless actual problems arise.

- **DATETIME=value:** (GDAL >= 3.10) Temporal filter. Its value should match the ones mentioned at paragraph Parameter datetime of the OGC API Features specification

7.56.7 Examples

7.56.7.1 Example 1: Listing the types of a OGC API - Features server

```
$ ogrinfo OAPIF:https://ogc-api.nrw.de/inspire-us-feuerwehr

INFO: Open of `OAPIF:https://ogc-api.nrw.de/inspire-us-feuerwehr'
      using driver `OAPIF' successful.
1: governmentalservice (title: Feuerwehrleitstellen) (Point)
```

7.56.7.2 Example 2: Listing the summary information of a OGC API - Features layer

```
$ ogrinfo OAPIF:https://ogc-api.nrw.de/inspire-us-feuerwehr governmentalservice -al -so

INFO: Open of `OAPIF:https://ogc-api.nrw.de/inspire-us-feuerwehr'
      using driver `OAPIF' successful.

Layer name: governmentalservice
Metadata:
  DESCRIPTION=Staatliche Verwaltungs- und Sozialdienste wie öffentliche Verwaltung,␣
↪Katastrophenschutz, Schulen und Krankenhäuser, die von öffentlichen oder privaten␣
↪Einrichtungen erbracht werden, soweit sie in den Anwendungsbereich der Richtlinie 2007/
↪2/EG fallen. Dieser Datensatz enthält Informationen zu Feuerwehrleitstellen.
  TITLE=Feuerwehrleitstellen
Geometry: Point
Feature Count: 52
Extent: (6.020720, 50.654901) - (9.199363, 52.300806)
Layer SRS WKT:
GEOGCRS["WGS 84",
    DATUM["World Geodetic System 1984",
        ELLIPSOID["WGS 84",6378137,298.257223563,
            LENGTHUNIT["metre",1]]],
    PRIMEM["Greenwich",0,
        ANGLEUNIT["degree",0.0174532925199433]],
    CS[ellipsoidal,2],
        AXIS["geodetic latitude (Lat)",north,
            ORDER[1],
            ANGLEUNIT["degree",0.0174532925199433]],
        AXIS["geodetic longitude (Lon)",east,
            ORDER[2],
            ANGLEUNIT["degree",0.0174532925199433]],
    ID["EPSG",4326]]
Data axis to CRS axis mapping: 2,1
id: String (0.0)
name: String (0.0)
inspireId: String (0.0)
serviceType.title: String (0.0)
```

(continues on next page)

(continued from previous page)

```
serviceType.href: String (0.0)
areaOfResponsibility.1.title: String (0.0)
areaOfResponsibility.1.href: String (0.0)
pointOfContact.address.thoroughfare: String (0.0)
pointOfContact.address.locatorDesignator: String (0.0)
pointOfContact.address.postCode: String (0.0)
pointOfContact.address.adminUnit: String (0.0)
pointOfContact.address.text: String (0.0)
pointOfContact.telephoneVoice: String (0.0)
pointOfContact.telephoneFacsimile: String (0.0)
pointOfContact.telephoneFacsimileEmergency: String (0.0)
inDistrict.title: String (0.0)
inDistrict.href: String (0.0)
inDistrictFreeTown.title: String (0.0)
inDistrictFreeTown.href: String (0.0)
inGovernmentalDistrict.title: String (0.0)
inGovernmentalDistrict.href: String (0.0)
```

7.56.7.3 Example 3: Filtering on a property

Depending on if the server exposes filtering capabilities of the properties, part or totally of the filter might be evaluated on client side.

```
$ ogrinfo OAPIF:https://ogc-api.nrw.de/inspire-us-feuerwehr governmentalservice -al -q -
↪where "name = 'Schwelm'"

Layer name: governmentalservice
Metadata:
  DESCRIPTION=Staatliche Verwaltungs- und Sozialdienste wie öffentliche Verwaltung,␣
↪Katastrophenschutz, Schulen und Krankenhäuser, die von öffentlichen oder privaten␣
↪Einrichtungen erbracht werden, soweit sie in den Anwendungsbereich der Richtlinie 2007/
↪2/EG fallen. Dieser Datensatz enthält Informationen zu Feuerwehrleitstellen.
  TITLE=Feuerwehrleitstellen
OGRFeature(governmentalservice):1
  id (String) = LtS01
  name (String) = Schwelm
  inspireId (String) = https://geodaten.nrw.de/id/inspire-us-feuerwehr/
↪governmentalservice/LtS01
  serviceType.title (String) = Brandschutzdienst
  serviceType.href (String) = http://inspire.ec.europa.eu/codelist/ServiceTypeValue/
↪fireProtectionService
  areaOfResponsibility.1.title (String) = Breckerfeld
  areaOfResponsibility.1.href (String) = https://registry.gdi-de.org/id/de.nw.inspire.au.
↪basis-dlm/AdministrativeUnit_05954004
  pointOfContact.address.thoroughfare (String) = Hauptstr.
  pointOfContact.address.locatorDesignator (String) = 92
  pointOfContact.address.postCode (String) = 58332
  pointOfContact.address.adminUnit (String) = Schwelm
  pointOfContact.address.text (String) = Hauptstr. 92, 58332 Schwelm
  pointOfContact.telephoneVoice (String) = +49233644400
  pointOfContact.telephoneFacsimile (String) = +4923364440400
  pointOfContact.telephoneFacsimileEmergency (String) = +49233644407100
```

(continues on next page)

(continued from previous page)

```
  inDistrict.title (String) = Ennepe-Ruhr
  inDistrict.href (String) = Ennepe-Ruhr
  inGovernmentalDistrict.title (String) = Arnsberg
  inGovernmentalDistrict.href (String) = https://registry.gdi-de.org/id/de.nw.inspire.au.
↪basis-dlm/AdministrativeUnit_059
  POINT (7.29854802787082 51.2855116825595)
```

7.56.7.4 Example 4: Spatial filtering

```
$ ogrinfo OAPIF:https://ogc-api.nrw.de/inspire-us-feuerwehr governmentalservice -al -q -
↪spat 7.1 51.2 7.2 51.5

Layer name: governmentalservice
Metadata:
  DESCRIPTION=Staatliche Verwaltungs- und Sozialdienste wie öffentliche Verwaltung,␣
↪Katastrophenschutz, Schulen und Krankenhäuser, die von öffentlichen oder privaten␣
↪Einrichtungen erbracht werden, soweit sie in den Anwendungsbereich der Richtlinie 2007/
↪2/EG fallen. Dieser Datensatz enthält Informationen zu Feuerwehrleitstellen.
  TITLE=Feuerwehrleitstellen
OGRFeature(governmentalservice):1
  id (String) = LtS33
  name (String) = Wuppertal-Solingen
  inspireId (String) = https://geodaten.nrw.de/id/inspire-us-feuerwehr/
↪governmentalservice/LtS33
  serviceType.title (String) = Brandschutzdienst
  serviceType.href (String) = http://inspire.ec.europa.eu/codelist/ServiceTypeValue/
↪fireProtectionService
  areaOfResponsibility.1.title (String) = Wuppertal
  areaOfResponsibility.1.href (String) = https://registry.gdi-de.org/id/de.nw.inspire.au.
↪basis-dlm/AdministrativeUnit_05124000
  pointOfContact.address.thoroughfare (String) = August-Bebel-Str.
  pointOfContact.address.locatorDesignator (String) = 55
  pointOfContact.address.postCode (String) = 42109
  pointOfContact.address.adminUnit (String) = Wuppertal
  pointOfContact.address.text (String) = August-Bebel-Str. 55, 42109 Wuppertal
  pointOfContact.telephoneVoice (String) = +492025631111
  pointOfContact.telephoneFacsimile (String) = +49202445331
  pointOfContact.telephoneFacsimileEmergency (String) = 112
  inDistrictFreeTown.title (String) = Wuppertal
  inDistrictFreeTown.href (String) = Wuppertal
  inGovernmentalDistrict.title (String) = Düsseldorf
  inGovernmentalDistrict.href (String) = https://registry.gdi-de.org/id/de.nw.inspire.au.
↪basis-dlm/AdministrativeUnit_051
  POINT (7.13806554104892 51.2674471939457)
```

7.56.8 See Also

- "OGC API - Features - Part 1: Core" Standard
- "OGC API - Features - Part 2: Coordinate Reference Systems by Reference" Standard
- *WFS (1.0,1.1,2.0) driver documentation*

7.57 Oracle Spatial

Driver short name

OCI

Build dependencies

OCI library

This driver supports reading and writing data in Oracle Spatial (8.1.7 or later) Object-Relational format. The Oracle Spatial driver is not normally built into OGR, but may be built in on platforms where the Oracle client libraries are available.

When opening a database, its name should be specified in the form "OCI:userid/password@database_instance:table,table". The list of tables is optional. The database_instance portion may be omitted when accessing the default local database instance. See the Oracle Help Center for more information about the OCI driver.

If the list of tables is not provided, then all tables appearing in ALL_SDO_GEOM_METADATA will be treated by OGR as layers with the table names as the layer names. Non-spatial tables or spatial tables not listed in the ALL_SDO_GEOM_METADATA view[1] are not accessible unless explicitly listed in the datasource name. Even in databases where all desired layers are in the ALL_SDO_GEOM_METADATA view, it may be desirable to list only the tables to be used as this can substantially reduce initialization time in databases with many tables.

If the table has an integer column called OGR_FID it will be used as the feature id by OGR (and it will not appear as a regular attribute). When loading data into Oracle Spatial OGR will always create the OGR_FID field.

When reading data from one or more views, the view names should be specified in the form "OCI:userid/password@database_instance:view,view". What is written above regarding tables, applies to views as well.

7.57.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

7.57.2 SQL Issues

By default, the Oracle driver passes SQL statements directly to Oracle rather than evaluating them internally when using the ExecuteSQL() call on the OGRDataSource, or the -sql command option to ogr2ogr. Attribute query expressions are also passed through to Oracle.

[1] It is the database user that is responsible for updating the ALL_SDO_GEOM_METADATA view, by inserting an appropriate row into the USER_SDO_GEOM_METADATA view. This is why it is possible that the table you want to read from is not listed in the ALL_SDO_GEOM_METADATA.

As well two special commands are supported via the ExecuteSQL() interface. These are "**DELLAYER:<table_name>**" to delete a layer, and "**VALLAYER:<table_name>**" to apply the SDO_GEOM.VALIDATE_GEOMETRY() check to a layer. Internally these pseudo-commands are translated into more complex SQL commands for Oracle.

It is also possible to request the driver to handle SQL commands with *OGR SQL* engine, by passing **"OGRSQL"** string to the ExecuteSQL() method, as name of the SQL dialect.

7.57.3 Caveats

- The type recognition logic is currently somewhat impoverished. No effort is made to preserve real width information for integer and real fields.
- Various types such as objects, and BLOBs in Oracle will be completely ignored by OGR.
- Currently the OGR transaction semantics are not properly mapped onto transaction semantics in Oracle.
- If an attribute called OGR_FID exists in the schema for tables being read, it will be used as the FID. Random (FID based) reads on tables without an identified (and indexed) FID field can be very slow. To force use of a particular field name the *OCI_FID* configuration option (e.g. environment variable) can be set to the target field name.
- Curved geometry types are converted to linestrings or linear rings in six degree segments when reading. The driver has no support for writing curved geometries.
- There is no support for point cloud (SDO_PC), TIN (SDO_TIN) and annotation text data types in Oracle Spatial.
- It might be necessary to define the environment variable NLS_LANG to "American_America.UTF8" to avoid issues with floating point numbers being truncated to integer on non-English environments.
- For developers: when running the driver under the memory error detection tool Valgrind, specifying the database_instance, typically to localhost, or with the TWO_TASK environment variable seems to be compulsory, otherwise "TNS:permission denied" errors will be reported)
- The logic for finding the specified table or view first checks whether a table with the given name exists, then a view, and then tries again with quoted names. This may result in one or more errors of the following type written to the output: "ORA-04043: object <object_name> does not exist", even if the object actually is found later on.

7.57.4 Creation Issues

The Oracle Spatial driver does not support creation of new datasets (database instances), but it does allow creation of new layers within an existing database.

Upon closing the OGRDataSource newly created layers will have a spatial index automatically built. At this point the USER_SDO_GEOM_METADATA table will also be updated with bounds for the table based on the features that have actually been written. One consequence of this is that once a layer has been loaded it is generally not possible to load additional features outside the original extents without manually modifying the DIMINFO information in USER_SDO_GEOM_METADATA and rebuilding the spatial index.

7.57.5 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The following configuration options are available:

- **OCI_FID=value:** Defaults to `OGR_FID`. Sets the name of the field to be used as the FID.
- **OCI_PREFETCH_ROWS=value:** (GDAL >= 3.14) Defaults to `1000`. Sets the number of rows to prefetch per round-trip.

7.57.5.1 Layer Creation Options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALDatasetCreateLayer()`* (C) or *`Dataset.CreateLayer`* (Python). The following layer creation options are supported:

- **OVERWRITE=[YES/NO]:** Defaults to `NO`. This may be "YES" to force an existing layer (=table) of the same desired name to be destroyed before creating the requested layer.
- **TRUNCATE=[YES/NO]:** Defaults to `NO`. This may be "YES" to force the existing table to be reused, but to first truncate all records in the table, preserving indexes or dependencies.
- **LAUNDER=[YES/NO]:** Defaults to `NO`. This may be "YES" to force new fields created on this layer to have their field names "laundered" into a form more compatible with Oracle. This converts to upper case and converts some special characters like "-" and "#" to "_".
- **PRECISION=[YES/NO]:** Defaults to `YES`. This may be "YES" to force new fields created on this layer to try and represent the width and precision information, if available using NUMBER(width,precision) or VARCHAR2(width) types. If "NO" then the types NUMBER, INTEGER and VARCHAR2 will be used instead.
- **DIM=[2/3]:** This may be set to 2 or 3 to force the dimension of the created layer. The dimension of the layer geometry type is used by default.
- **SPATIAL_INDEX=[YES/NO]:** This may be set to NO to disable creation of a spatial index when a layer load is complete. By default an index is created if any of the layer features have valid geometries. The default is "YES".
- **INDEX_PARAMETERS=value:** This may be set to pass creation parameters when the spatial index is created. For instance setting *`INDEX_PARAMETERS`* to `SDO_RTR_PCTFREE=0` would cause the rtree index to be created without any empty space. By default no parameters are passed causing a default R-Tree spatial index to be created.
- **ADD_LAYER_GTYPE=[YES/NO]:** Defaults to `YES`. This may be set to NO to disable the constraints on the geometry type in the spatial index, through the layer_gtype keyword in the PARAMETERS clause of the CREATE INDEX. Layers of type MultiPoint, MultiLineString or MultiPolygon will also accept single geometry type (Point, LineString, Polygon).
- **DIMINFO_X=value:** This may be set to xmin,xmax,xres values to control the X dimension info written into the USER_SDO_GEOM_METADATA table. By default extents are collected from the actual data written.
- **DIMINFO_Y=value:** This may be set to ymin,ymax,yres values to control the Y dimension info written into the USER_SDO_GEOM_METADATA table. By default extents are collected from the actual data written.
- **DIMINFO_Z=value:** This may be set to zmin,zmax,zres values to control the Z dimension info written into the USER_SDO_GEOM_METADATA table. By default fixed values of -100000,100000,0.002 are used for layers with a third dimension.
- **SRID=value:** By default this driver will attempt to find an existing row in the MDSYS.CS_SRS table with a well known text coordinate system exactly matching the one for this dataset. If one is not found, a new row will be added to this table. The SRID creation option allows the user to force use of an existing Oracle SRID item even it if does not exactly match the WKT the driver expects.
- **MULTI_LOAD=[YES/NO]:** Defaults to `YES`. If enabled new features will be created in groups of 100 per SQL INSERT command, instead of each feature being a separate INSERT command. Having this enabled is the fastest way to load data quickly. Multi-load mode is enabled by default, and may be forced off for existing layers or for new layers by setting to NO. The number of rows in each group is defined by MULTI_LOAD_COUNT. To load one row at a time, set MULTI_LOAD to NO.
- **MULTI_LOAD_COUNT=value:** Define the number of features on each ARRAY INSERT command, instead of the default 100 item defined by *`MULTI_LOAD`*. Since each array insert will commit a transaction, this options shouldn't be combined with ogr2ogr "-gt N". Use "-gt unlimited" preferably when using

MULTI_LOAD_COUNT. The default is 100. If neither *MULTI_LOAD* nor *MULTI_LOAD_COUNT* are specified, then the loading happens in groups of 100 rows.

- **FIRST_ID=value:** Define the first numeric value of the id column on the first rows. It also works as a open option when used to append or update an existing dataset.
- **NO_LOGGING=[YES/NO]:** Define that the table and the geometry will be create with nologging attributes.
- **LOADER_FILE=value:** If this option is set, all feature information will be written to a file suitable for use with SQL*Loader instead of inserted directly in the database. The layer itself is still created in the database immediately. The SQL*Loader support is experimental, and generally *MULTI_LOAD* enabled mode should be used instead when trying for optimal load performance.
- **GEOMETRY_NAME=value:** Defaults to `ORA_GEOMETRY`. By default OGR creates new tables with the geometry column named ORA_GEOMETRY. If you wish to use a different name, it can be supplied with the GEOMETRY_NAME layer creation option.
- **TIMESTAMP_WITH_TIME_ZONE=[YES/NO]:** (GDAL >= 3.10.1) Defaults to `NO`. Whether DateTime fields should be created with TIMESTAMP WITH TIME ZONE Oracle type (otherwise without timezone). When creating a field, if it advertises a known or mixed time zone, TIMESTAMP_WITH_TIME_ZONE will default to YES, otherwise it will default to NO.

7.57.5.2 Layer Open Options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are supported:

- **FIRST_ID=value:** See Layer Create Options comments on *FIRST_ID*.
- **MULTI_LOAD=value:** See Layer Create Options comments on *MULTI_LOAD*.
- **MULTI_LOAD_COUNT=value:** See Layer Create Options comments on :lco:MULTI_LOAD_COUNT`.
- **WORKSPACE=value:** Define what user workspace to use.

7.57.6 Examples

7.57.6.1 Example 1

Simple translation of a shapefile into Oracle. The table 'ABC' will be created with the features from abc.shp and attributes from abc.dbf.

```
ogr2ogr -f OCI OCI:warmerda/password@gdal800.dreadfest.com abc.shp
```

7.57.6.2 Example 2

This example shows using ogrinfo to evaluate an SQL query statement within Oracle. More sophisticated Oracle Spatial specific queries may also be used via the -sql commandline switch to ogrinfo.

```
ogrinfo -ro OCI:warmerda/password -sql "SELECT pop_1994 from canada where province_name␣
↪= 'Alberta'"
```

7.57.6.3 Example 3

This example shows hows to list information about an Oracle view.

```
ogrinfo -ro -so OCI:username/password@host_name:port_number/service_name:MY_SCHEMA.MY_
↪VIEW MY_SCHEMA.MY_VIEW
```

7.57.6.4 Example 4

This example shows hows to convert certain columns from an Oracle view to a GeoPackage file, explicitly assigning the layer name and the coordinate reference system, and converting timestamps to UTC.

```
ogr2ogr -f GPKG output.gpkg -nln new_layer_name -nlt POLYGON -s_srs EPSG:25832 -t_srs␣
↪EPSG:25832 -dsco DATETIME_FORMAT=UTC OCI:username/password@host_name:port_number/
↪service_name:MY_SCHEMA.MY_VIEW -sql "SELECT COLUMN_A, COLUMN_B, GEOMETRY FROM MY_
↪SCHEMA.MY_VIEW"
```

7.57.6.5 Standalone plugin compilation

Added in version 3.10.

While this driver may be built as part of a whole GDAL build, either in libgdal itself, or as a plugin, it is also possible to only build this driver as a plugin, against an already built libgdal.

The version of the GDAL sources used to build the driver must match the version of the libgdal it is built against.

For example, from a "build_oci" directory under the root of the GDAL source tree:

```
cmake -S ../ogr/ogrsf_frmts/oci -DCMAKE_PREFIX_PATH=/path/to/GDAL_installation_prefix -
↪DOracle_ROOT=/path/to/instantclient_sdk_root
cmake --build .
```

Note that such a plugin, when used against a libgdal not aware of it, will be systematically loaded at GDAL driver initialization time, and will not benefit from rfc-96. For that, libgdal itself must be built with the CMake variable OGR_REGISTER_DRIVER_OCI_FOR_LATER_PLUGIN=ON set.

7.57.6.6 Credits

SRC, LLC for its financial support of the initial development of this driver.

7.58 ODBC RDBMS

Driver short name

ODBC

Build dependencies

ODBC library

OGR optionally supports spatial and non-spatial tables accessed via ODBC. ODBC is a generic access layer for access to many database systems, and data that can be represented as a database (collection of tables). ODBC support is potentially available on Unix and Windows platforms, but is only included in unix builds by special configuration options.

ODBC datasources are accessed using a datasource name of the form **ODBC:userid/password@dsn,schema.tablename(geometrycolna**
With optional items dropped the following are also acceptable:

- **ODBC:userid/password@dsn**
- **ODBC:userid@dsn,table_list**

- **ODBC:dsn,table_list**
- **ODBC:dsn**
- **ODBC:dsn,table_list:srs_tablename**

The **dsn** is the ODBC Data Source Name. Normally ODBC datasources are setup using an ODBC Administration tool, and assigned a DSN. That DSN is what is used to access the datasource.

By default the ODBC searches for GEOMETRY_COLUMNS table. If found it is used to identify the set of spatial tables that should be treated as layers by OGR. If not found, then all tables in the datasource are returned as non-spatial layers. However, if a table list (a list of comma separated table names) is provided, then only those tables will be represented as layers (non-spatial). Fetching the full definition of all tables in a complicated database can be quite time consuming, so the ability to restrict the set of tables accessed is primarily a performance issue.

If the GEOMETRY_COLUMNS table is found, it is used to select a column to be the geometry source. If the tables are passed in the datasource name, then the geometry column associated with a table can be included in round brackets after the tablename. It is currently a hardcoded assumption that the geometry is in Well Known Binary (WKB) format if the field is binary, or Well Known Text (WKT) otherwise. The GEOMETRY_COLUMNS table should have at least the columns F_TABLE_NAME, F_GEOMETRY_COLUMN and GEOMETRY_TYPE.

If the table has a geometry column, and has fields called XMIN, YMIN, XMAX and YMAX then direct table queries with a spatial filter accelerate the spatial query. The XMIN, YMIN, XMAX and YMAX fields should represent the extent of the geometry in the row in the tables coordinate system.

By default, SQL statements are passed directly to the underlying database engine. It's also possible to request the driver to handle SQL commands with the *OGR SQL* engine, by passing **"OGRSQL"** string to the ExecuteSQL() method, as name of the SQL dialect.

7.58.1 Dataset open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are supported:

- **LIST_ALL_TABLES=[YES/NO]:** This may be "YES" to force all tables, including system and internal tables (such as the MSys* tables) to be listed (since GDAL 3.4). Applies to Microsoft Access Databases only. Note that the Windows Microsoft Access ODBC Driver always strips out MSys tables, and accordingly these will not be returned on Windows platforms even if LIST_ALL_TABLES is set to YES.

7.58.2 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

7.58.3 Access Databases (.MDB) support

On Windows provided that the "Microsoft Access Driver (*.mdb)" ODBC driver is installed, non-spatial MS Access Databases (not Personal Geodabases or Geomedia databases) can be opened directly by their filenames.

On Linux opening non-spatial MS Access Databases using the ODBC driver is possible via installation of unixODBC and mdbtools. See *MDB* for instructions on how to enable this.

The driver supports either .mdb or .accdb extensions for Microsoft Access databases. Additionally, it also supports opening files with the ESRI .style database extension (which is just an alias for the .mdb file extension).

7.58.4 Creation Issues

Currently the ODBC OGR driver is read-only, so new features, tables and datasources cannot normally be created by OGR applications. This limitation may be removed in the future.

7.58.4.1 See Also

- MSDN ODBC API Reference
- *PGeo driver*

7.59 ODS - Open Document Spreadsheet

Driver short name

ODS

Build dependencies

libexpat

This driver can read, write and update spreadsheets in Open Document Spreadsheet format, used by applications like OpenOffice / LibreOffice / KSpread / etc...

The driver is only available if GDAL/OGR is compiled against the Expat library.

Each sheet is presented as a OGR layer. No geometry support is available directly (but you may use the OGR VRT capabilities for that).

Note 1 : spreadsheets with passwords are not supported.

Note 2 : when updating an existing document, all existing styles, formatting, formulas and other concepts (charts, drawings, macros, ...) not understood by OGR will be lost : the document is re-written from scratch from the OGR data model.

7.59.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.59.2 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are available:

- **HEADERS=[FORCE/DISABLE/AUTO]:** (GDAL >= 3.8) Defaults to `AUTO`. By default, the driver will read the first lines of each sheet to detect if the first line might be the name of columns. If set to FORCE, the driver will

consider the first line as the header line. If set to DISABLE, it will be considered as the first feature. Otherwise auto-detection will occur.

- **FIELD_TYPES=[STRING/AUTO]:** (GDAL >= 3.8) Defaults to `AUTO`. By default, the driver will try to detect the data type of fields. If set to STRING, all fields will be of String type.

7.59.3 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The following configuration options are available:

- **OGR_ODS_HEADERS=[FORCE/DISABLE/AUTO]:** Defaults to `AUTO`. By default, the driver will read the first lines of each sheet to detect if the first line might be the name of columns. If set to FORCE, the driver will consider the first line as the header line. If set to DISABLE, it will be considered as the first feature. Otherwise auto-detection will occur.
- **OGR_ODS_FIELD_TYPES=[STRING/AUTO]:** Defaults to `AUTO`. By default, the driver will try to detect the data type of fields. If set to STRING, all fields will be of String type.

7.60 ESRI File Geodatabase vector (OpenFileGDB)

Driver short name

OpenFileGDB

Driver built-in by default

This driver is built-in by default

The OpenFileGDB driver provides read, write and update access to vector layers of File Geodatabases (.gdb directories) created by ArcGIS 10 and above (it has also read-only support for ArcGIS 9.x geodatabases). The dataset name must be the directory/folder name, and it must end with the .gdb extension.

It can also read directly zipped .gdb directories (with .gdb.zip extension), provided they contain a .gdb directory at their first level.

A specific .gdbtable file (including "system" tables) can also be opened directly.

Curve in geometries are supported.

Write and update capabilities are supported since GDAL >= 3.6

The driver also supports *raster layers* since GDAL 3.7

7.60.1 Driver capabilities

Supports Create()

Added in version GDAL: 3.6

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.60.2 Spatial filtering

Since GDAL 3.2, the driver can use the native .spx spatial indices for spatial filtering.

In earlier versions, it uses the minimum bounding rectangle included at the beginning of the geometry blobs to speed up spatial filtering. By default, it also builds on the fly a in-memory spatial index during the first sequential read of a layer. Following spatial filtering operations on that layer will then benefit from that spatial index. The building of this in-memory spatial index can be disabled by setting the `OPENFILEGDB_IN_MEMORY_SPI` configuration option to NO.

7.60.3 SQL support

SQL statements are run through the OGR SQL engine. When attribute indexes (.atx files) exist, the driver will use them to speed up WHERE clauses or SetAttributeFilter() calls.

7.60.3.1 Special SQL requests

"GetLayerDefinition a_layer_name" and "GetLayerMetadata a_layer_name" can be used as special SQL requests to get respectively the definition and metadata of a FileGDB table as XML content (only available in Geodatabases created with ArcGIS 10 or above)

The "CREATE INDEX idx_name ON layer_name(field_name)" SQL request can be used to create an attribute index. idx_name must have 16 characters or less, start with a letter and contain only alphanumeric characters or underscore. Multiple column indices are not supported.

The "RECOMPUTE EXTENT ON layer_name" SQL request can be used to trigger an update of the layer extent in layer metadata. This is useful when updating or deleting features that modify the general layer extent.

The "REPACK" or "REPACK layer_name" SQL requests can be used respectively to compact the whole database or a given layer. This is useful when doing editions (updates or feature deletions) that may leave holes in .gdbtable files. The REPACK command causes the .gdbtable to be rewritten without holes. Note that compaction does not involve extent recomputation.

7.60.4 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as `CPLSetConfigOption()` (C) or `gdal.config_options` (Python). The following configuration options are available:

- **OPENFILEGDB_IN_MEMORY_SPI=[YES/NO]:** If `YES`, an in-memory spatial index will be built instead of using the native spatial index. See *Spatial filtering*.
- **OPENFILEGDB_DEFAULT_STRING_WIDTH=<integer>:** Width of string fields to use on creation, when the width specified to CreateField() is the unspecified value 0. This defaults to 65536.

7.60.5 Dataset open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are supported:

- **LIST_ALL_TABLES=[YES/NO]:** (GDAL >= 3.4) This may be "YES" to force all tables, including system and internal tables (such as the GDB_* tables) to be listed

7.60.6 Dataset Creation Options

None.

7.60.7 Layer Creation Options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALDatasetCreateLayer()* (C) or *Dataset.CreateLayer* (Python). The following layer creation options are supported:

- **TARGET_ARCGIS_VERSION=[ALL/ARCGIS_PRO_3_2_OR_LATER]:** (GDAL >= 3.9) Defaults to `ALL`. ArcGIS version that the dataset must be compatible with. If creation of Integer64, Date, Time field types is needed, the `ARCGIS_PRO_3_2_OR_LATER` must be selected. If set or let to the default value `ALL`, those types will be respectively be encoded to fallback types (Float64 instead of Integer64, DateTime instead of Date or Time).
- **FEATURE_DATASET=<string>:** When this option is set, the new layer will be created inside the named FeatureDataset folder. If the folder does not already exist, it will be created.
- **LAYER_ALIAS=value:** Set layer name alias. On reading, this information is exposed in the ALIAS_NAME layer metadata item (GDAL >= 3.8).
- **GEOMETRY_NAME=value:** Defaults to `SHAPE`. Set name of geometry column in new layer.
- **GEOMETRY_NULLABLE=[YES/NO]:** Defaults to `YES`. Whether the values of the geometry column can be NULL. Can be set to NO so that geometry is required.
- **FID=value:** Defaults to `OBJECTID`. Name of the OID column to create.
- **XYTOLERANCE, ZTOLERANCE, MTOLERANCE**=value: These parameters control the snapping tolerance used for advanced ArcGIS features like network and topology rules. They won't effect any OGR operations, but they will by used by ArcGIS. The units of the parameters are the units of the coordinate reference system.

  ArcMap 10.0 and OGR defaults for XYTOLERANCE are 0.001m (or equivalent) for projected coordinate systems, and 0.000000008983153° for geographic coordinate systems. ArcMap 10.0 and OGR defaults for ZTOLERANCE and MTOLERANCE are 0.0001.
- **XORIGIN, YORIGIN, ZORIGIN, MORIGIN, XYSCALE, ZSCALE, ZORIGIN**=value: These parameters control the coordinate precision grid inside the file geodatabase. The dimensions of the grid are determined by the origin, and the scale. The origin defines the location of a reference grid point in space. The scale is the reciprocal of the resolution. So, to get a grid with an origin at 0 and a resolution of 0.001 on all axes, you would set all the origins to 0 and all the scales to 1000.

  Important: The domain specified by `(xmin=XORIGIN, ymin=YORIGIN, xmax=(XORIGIN + 9E+15 / XYSCALE), ymax=(YORIGIN + 9E+15 / XYSCALE))` needs to encompass every possible coordinate value for the feature class. If features are added with coordinates that fall outside the domain, errors will occur in ArcGIS with spatial indexing, feature selection, and exporting data.

  ArcMap 10.0 and OGR defaults:

  - For geographic coordinate systems: XORIGIN=-400, YORIGIN=-400, XYSCALE=1000000000
  - For projected coordinate systems: XYSCALE=10000 for the default XYTOLERANCE of 0.001m. XORIGIN and YORIGIN change based on the coordinate system, but the OGR default of -2147483647 is suitable with the default XYSCALE for all coordinate systems.

  - ZORIGIN and MORIGIN: -100000
  - ZSCALE and MSCALE: 10000

- **COLUMN_TYPES=value:** A list of strings of format field_name=fgdb_field_type (separated by comma) to force the FileGDB column type of fields to be created.
- **DOCUMENTATION=<string>:** XML documentation for the layer.
- **CONFIGURATION_KEYWORD=[DEFAULTS/MAX_FILE_SIZE_4GB/MAX_FILE_SIZE_256TB]:** Customize how data is stored. By default text in UTF-8 and data up to 1TB
- **CREATE_SHAPE_AREA_AND_LENGTH_FIELDS=[YES/NO]:** Defaults to `NO`. When this option is set, a Shape_Area and Shape_Length special fields will be created for polygonal layers (Shape_Length only for linear layers). These fields will automatically be populated with the feature's area or length whenever a new feature is added to the dataset or an existing feature is amended. When using ogr2ogr with a source layer that has Shape_Area/Shape_Length special fields, and this option is not explicitly specified, it will be automatically set, so that the resulting FileGeodatabase has those fields properly tagged.

7.60.8 64-bit integer field support

Added in version 3.9.

On creation, 64-bit integer field support requires setting the *`TARGET_ARCGIS_VERSION`* layer creation option to `ARCGIS_PRO_3_2_OR_LATER`. Note that Esri recommends to restrict the range of 64-bit integer values to [-9007199254740991, 9007199254740991] for the larger compatibility. GDAL will allow writing values outside of that range without warning, and can also read them fine.

7.60.9 Field domains

Added in version 3.3.

Coded and range field domains are supported.

7.60.10 Relationships

Added in version 3.6.

Relationship retrieval, creation, deletion and updating is supported.

7.60.11 Hierarchical organization

Added in version 3.4.

The hierarchical organization of tables and feature classes as top-level element or within a feature dataset can be explored using the methods *`GDALDataset::GetRootGroup()`*, *`GDALGroup::GetGroupNames()`*, *`GDALGroup::OpenGroup()`*, *`GDALGroup::GetVectorLayerNames()`* and *`GDALGroup::OpenVectorLayer()`*

7.60.12 Transaction support

The driver implements transactions at the database level, through an emulation (as per rfc-54). This works by backing up the current state of the modified parts of a geodatabase after StartTransaction(force=TRUE) is called. If the transaction is committed, the backup copy is destroyed. If the transaction is rolled back, the backup copy is restored.

Note that this emulation has an unspecified behavior in case of concurrent updates (with different connections in the same or another process).

7.60.13 Geometry coordinate precision

Added in version GDAL: 3.9

The driver supports reading and writing the geometry coordinate precision, using the XYResolution, ZResolution and MResolution members of the *OGRGeomCoordinatePrecision* settings of the *OGRGeomFieldDefn*. `XYScale` is computed as 1.0 / `XYResolution` (and similarly for the Z and M components). The tolerance setting is computed as being one tenth of the resolution

On reading, the coordinate precision grid parameters are returned as format specific options of *OGRGeomCoordinatePrecision* with the `FileGeodatabase` format key, with the following option key names: `XYScale`, `XYTolerance`, `XYOrigin`, `ZScale`, `ZTolerance`, `ZOrigin`, `MScale`, `MTolerance`, `MOrigin`. On writing, they are also honored (they will have precedence over XYResolution, ZResolution and MResolution).

On layer creation, the XORIGIN, YORIGIN, ZORIGIN, MORIGIN, XYSCALE, ZSCALE, ZORIGIN, XYTOLERANCE, ZTOLERANCE, MTOLERANCE layer creation options will be used in priority over the settings of *OGRGeomCoordinatePrecision*.

7.60.14 Advantages of the OpenFileGDB driver, compared to the FileGDB driver

- Can read ArcGIS 9.X Geodatabases, and not only 10 or above.
- Can open layers with any spatial reference system.
- Thread-safe (i.e. datasources can be processed in parallel).
- Uses the VSI Virtual File API, enabling the user to read a Geodatabase in a ZIP file or stored on a HTTP server.
- Faster on databases with a big number of fields.
- Does not depend on a third-party library.
- Robust against corrupted Geodatabase files.

7.60.15 Limitations

- Reading data compressed in SDC format (Smart Data Compression) or in CDF format (Compressed Data Format) is not supported. For CDF, the *FileGDB driver* can be used.
- Support for tables with 64-bit OBJECTIDs (which require GDAL >= 3.10), where OBJECTIDs are sparse, is read-only and incomplete. The driver will emit a warning if such situation occurs, and will attribute non-faithful OBJECTIDs.

7.60.16 Examples

7.60.16.1 Example 1: Reading a layer from FileGDB and load into PostGIS

```
ogr2ogr -overwrite -f "PostgreSQL" PG:"host=myhost user=myuser dbname=mydb␣
↪password=mypass" "C:\somefolder\BigFileGDB.gdb" "MyFeatureClass"
```

7.60.16.2 Example 2: Getting detailed info for FileGDB

```
ogrinfo -al "C:\somefolder\MyGDB.gdb"
```

7.60.16.3 Example 3: Getting detailed info for a zipped FileGDB

```
ogrinfo -al "C:\somefolder\MyGDB.gdb.zip"
```

7.60.17 Links

- *OpenFileGDB raster* documentation page
- *FileGDB driver*, relying on the FileGDB API SDK
- Reverse-engineered specification of the FileGDB format

7.60.18 Credits

Edition/write capabilities of the driver have been funded by the following organizations: Provincie Zuid-Holland, Provincie Gelderland and Gemeente Amsterdam.

7.61 OSM - OpenStreetMap XML and PBF

Driver short name

OSM

Build dependencies

libsqlite3 (and libexpat for OSM XML)

This driver reads OpenStreetMap files, in .osm (XML based) and .pbf (optimized binary) formats.

The driver reconstructs point, line and polygon geometries from the OpenStreetMap base objects, which are nodes, ways and relations. Such operation can be very slow for large files. Consequently, except for small extracts of the OSM database (that is .pbf files of a few megabytes at most), using this driver for interactive reading is not appropriate. It is better used to import an OSM file into a proper GIS format such as GeoPackage. Even if the driver can theoretically deal with arbitrarily large OSM files, it is not recommended to use it to import the whole planet. A tool such as osm2pgsql which imports into a PostgreSQL database is much better suited for such operation.

The driver is available if GDAL is built with SQLite support and, for .osm XML files, with Expat support.

The filenames must end with .osm or .pbf extension.

The driver will categorize features into 5 layers :

- **points** : "node" features that have significant tags attached.
- **lines** : "way" features that are recognized as non-area.
- **multilinestrings** : "relation" features that form a multilinestring(type = 'multilinestring' or type = 'route').
- **multipolygons** : "relation" features that form a multipolygon (type = 'multipolygon' or type = 'boundary'), and "way" features that are recognized as area.
- **other_relations** : "relation" features that do not belong to the above 2 layers.

7.61.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.61.2 Configuration

In the *data* folder of the GDAL distribution, you can find an osmconf.ini file that can be customized to fit your needs. You can also define an alternate path with the *OSM_CONFIG_FILE* configuration option.

The customization is essentially which OSM attributes and keys should be translated into OGR layer fields.

Fields can be computed with SQL expressions (evaluated by SQLite engine) from other fields/tags. For example to compute the z_order attribute.

7.61.2.1 "other_tags" field

When keys are not strictly identified in the *osmconf.ini* file, the key/value pair is appended in a "other_tags" field, with a syntax compatible with the PostgreSQL HSTORE type. See the *COLUMN_TYPES* layer creation option of the *PG driver*. The `hstore_get_value()` function can be used with the *OGRSQL* or *SQLite* SQL dialects to extract the value of a given key.

For example :

```
ogr2ogr -f PostgreSQL "PG:dbname=osm" test.pbf \
-lco COLUMN_TYPES=other_tags=hstore
```

Starting with GDAL 3.7, it is possible to ask the format of this field to be JSON encoded, by using the TAGS_FORMAT=JSON open option, or the `tags_format=json` setting in the *osmconf.ini* file.

7.61.2.2 "all_tags" field

Similar to "other_tags", except that it contains both keys specifically identified to be reported as dedicated fields, as well as other keys.

"all_tags" is disabled by default, and when enabled, it is exclusive with "other_tags".

7.61.3 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). The following configuration options are available:

- **OSM_CONFIG_FILE=<filename>:** Path to an OSM configuration file. See *Configuration*.
- **OSM_MAX_TMPFILE_SIZE=<MB>:** Defaults to `100`. Size of the internal SQLite database used to resolve geometries. If that database remains under the size specified by this option, it will reside in RAM. If it grows above, it will be written in a temporary file on disk. By default, this file will be written in the current directory, unless you define the *CPL_TMPDIR* configuration option.
- **OSM_USE_CUSTOM_INDEXING=[YES/NO]:** Defaults to `YES`. For indexation of nodes, a custom mechanism not relying on SQLite is used by default (indexation of ways to solve relations is still relying on SQLite). It can speed up operations significantly. However, in some situations (non increasing node ids, or node ids not in expected range), it might not work and the driver will output an error message suggesting to relaunch by setting this configuration option to NO.
- **OSM_COMPRESS_NODES=[YES/NO]:** Defaults to `NO`. When custom indexing is used (*OSM_USE_CUSTOM_INDEXING=YES*, default case), the *OSM_COMPRESS_NODES* configuration option can be set to YES. This option might be turned on to improve performance when I/O access is the limiting factor

(typically the case of rotational disk), and will be mostly efficient for country-sized OSM extracts where compression rate can go up to a factor of 3 or 4, and help keep the node DB to a size that fit in the OS I/O caches. For whole planet file, the effect of this option will be less efficient. This option consumes additional 60 MB of RAM.

- **OGR_INTERLEAVED_READING=value:** See *Interleaved reading*.

7.61.4 Interleaved reading

Due to the nature of OSM files and how the driver works internally, the default reading mode that works per-layer might not work correctly, because too many features will accumulate in the layers before being consumed by the user application.

Applications should use the `GDALDataset::GetNextFeature()` API to iterate over features in the order they are produced.

For earlier versions, for large files, applications should set the *OGR_INTERLEAVED_READING=YES* configuration option to turn on a special reading mode where the following reading pattern must be used:

```
bool bHasLayersNonEmpty;
do
{
    bHasLayersNonEmpty = false;

    for( int iLayer = 0; iLayer < poDS->GetLayerCount(); iLayer++ )
    {
        OGRLayer *poLayer = poDS->GetLayer(iLayer);

        OGRFeature* poFeature;
        while( (poFeature = poLayer->GetNextFeature()) != NULL )
        {
            bHasLayersNonEmpty = true;
            OGRFeature::DestroyFeature(poFeature);
        }
    }
}
while( bHasLayersNonEmpty );
```

Note : the ogr2ogr application has been modified to use that *OGR_INTERLEAVED_READING* mode without any particular user action.

7.61.5 Spatial filtering

Due to way .osm or .pbf files are structured and the parsing of the file is done, for efficiency reasons, a spatial filter applied on the points layer will also affect other layers. This may result in lines or polygons that have missing vertices.

To improve this, a possibility is using a larger spatial filter with some buffer for the points layer, and then post-process the output to apply the desired filter. This would not work however if a polygon has vertices very far away from the interest area. In which case full conversion of the file to another format, and filtering of the resulting lines or polygons layers would be needed.

7.61.6 Reading .osm.bz2 files and/or online files

.osm.bz2 are not natively recognized, however you can process them (on Unix), with the following command :

```
bzcat my.osm.bz2 | ogr2ogr -f SQLite my.sqlite /vsistdin/
```

You can convert a .osm or .pbf file without downloading it :

```
wget -O - http://www.example.com/some.pbf |
ogr2ogr -f SQLite my.sqlite /vsistdin/

or

ogr2ogr -f SQLite my.sqlite \
/vsicurl_streaming/http://www.example.com/some.pbf -progress
```

And to combine the above steps :

```
wget -O - http://www.example.com/some.osm.bz2 |
bzcat | ogr2ogr -f SQLite my.sqlite /vsistdin/
```

7.61.7 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are supported:

- **CONFIG_FILE=<filename>:** Defaults to `{GDAL_DATA}/osmconf.ini`. Configuration filename.
- **USE_CUSTOM_INDEXING=[YES/NO]:** Defaults to `YES`. Whether to enable custom indexing.
- **COMPRESS_NODES=[YES/NO]:** Defaults to `NO`. Whether to compress nodes in temporary DB.
- **MAX_TMPFILE_SIZE=<MBytes>:** Defaults to `100`. Maximum size in MB of in-memory temporary file. If it exceeds that value, it will go to disk.
- **INTERLEAVED_READING=[YES/NO]:** Defaults to `NO`. Whether to enable interleaved reading.
- **TAGS_FORMAT=[HSTORE/JSON]:** (GDAL >= 3.7) Defaults to `HSTORE`. Format for all_tags/other_tags fields.

7.61.8 Examples

7.61.8.1 Example 1

List the contents of a local PBF file, with the original downloaded from https://download.geofabrik.de/europe/bosnia-herzegovina-latest.osm.pbf.

ogr2ogr

gdal vector info

```
ogrinfo bosnia-herzegovina-latest.osm.pbf
```

```
gdal vector info bosnia-herzegovina-latest.osm.pbf
```

7.61.8.2 Example 2

Export all points tagged as trees (`"natural"=>"tree"`) to a GeoJSON file:

Bash

PowerShell

```
ogr2ogr trees.geojson bosnia-herzegovina-latest.osm.pbf \
  -sql "SELECT * FROM points WHERE other_tags LIKE '%\"natural\"=>\"tree\"%'" \
  -dialect sqlite
```

```
ogr2ogr trees.geojson bosnia-herzegovina-latest.osm.pbf `
  -sql "SELECT * FROM points WHERE other_tags LIKE '%\""natural\""=>\""tree\""%'" `
  -dialect sqlite
```

7.61.8.3 Example 3

As above, but using a *gdal vector pipeline* and with an additional bounding box filter for the Mostar region:

Bash

PowerShell

```
gdal vector pipeline \
  ! read --input bosnia-herzegovina-latest.osm.pbf \
  ! filter --bbox 17.7789,43.3111,17.8379,43.3677 \
  ! sql "SELECT * FROM points WHERE other_tags LIKE '%\"natural\"=>\"tree\"%'" --dialect␣
↪sqlite \
  ! write trees-filtered.geojson
```

```
gdal vector pipeline `
  ! read --input bosnia-herzegovina-latest.osm.pbf `
  ! filter --bbox 17.7789,43.3111,17.8379,43.3677 `
  ! sql "SELECT * FROM points WHERE other_tags LIKE '%\""natural\""=>\""tree\""%'" --
↪dialect sqlite `
  ! write trees-filtered.geojson
```

7.61.9 See Also

- OpenStreetMap home page
- OSM XML Format description
- OSM PBF Format description

7.62 (Geo)Parquet

Added in version 3.5.

Driver short name

Parquet

Build dependencies

Parquet component of the Apache Arrow C++ library

From https://parquet.apache.org/: "Apache Parquet is an open source, column-oriented data file format designed for efficient data storage and retrieval. It provides efficient data compression and encoding schemes with enhanced performance to handle complex data in bulk. Parquet is available in multiple languages including Java, C++, Python, etc..."

This driver also supports geometry columns using the GeoParquet specification.

The GeoParquet 1.0.0 specification is supported since GDAL 3.8.0. The GeoParquet 1.1.0 specification is supported since GDAL 3.9.0.

7.62.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.62.2 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are supported:

- **GEOM_POSSIBLE_NAMES=value:** (GDAL >= 3.8) Defaults to `geometry,wkb_geometry,wkt_geometry`. Comma separated list of possible names for geometry column(s). Only used for files without GeoParquet dataset-level metadata. Columns are recognized as geometry columns only if they are of type binary (they are assumed to contain WKB encoded geometries), or if they are of type string and contain the "wkt" substring in their name (they are then assumed to contain WKT encoded geometries).
- **CRS=value:** (GDAL >= 3.8) To set or override the CRS of geometry columns. The string is typically formatted as CODE:AUTH (e.g "EPSG:4326"), or can be a PROJ.4 or WKT CRS string.
- **LISTS_AS_STRING_JSON=[YES/NO]:** (GDAL >= 3.12.1) Defaults to `NO`. Whether lists of strings/integers/reals should be reported as String(JSON) fields rather than String/Integer[64]/RealList. Useful when null values in such lists must be exactly mapped as such, instead of being omitted (for lists of strings), or set to 0 (for list of boolean, integer or real).

7.62.3 Creation issues

The driver supports creating only a single layer in a dataset.

7.62.4 Layer creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALDatasetCreateLayer()* (C) or *Dataset.CreateLayer* (Python). The following layer creation options are supported:

- **COMPRESSION=[NONE/UNCOMPRESSED/SNAPPY/GZIP/BROTLI/ZSTD/LZ4_RAW/LZ4_HADOOP]:** Compression method. Available values depend on how the Parquet library was compiled. Defaults to SNAPPY when available, otherwise NONE.
- **COMPRESSION_LEVEL=<integer>:** (GDAL >= 3.12) Specify the compression level for the selected compression method. The compression level has a different meaning for each codec. The description of this option, returned at runtime for example by `ogrinfo --format PARQUET`, gives the range and default value for each codec.
- **GEOMETRY_ENCODING=[WKB/WKT/GEOARROW/GEOARROW_INTERLEAVED]:** Defaults to `WKB`. Geometry encoding. WKB is the default and recommended choice for maximal interoperability. WKT is *not* allowed by the GeoParquet specification, but are handled as an extension. As of GDAL 3.9, GEOARROW uses the GeoParquet 1.1 GeoArrow "struct" based encodings (where points are modeled as a struct field with a x and y subfield, lines are modeled as a list of such points, etc.). The GEOARROW_INTERLEAVED option has been renamed in GDAL 3.9 from what was named GEOARROW in previous versions, and uses an encoding where points uses a FixedSizedList of (x,y), lines a variable-size list of such FixedSizedList of points, etc. This GEOARROW_INTERLEAVED encoding is not part of the official GeoParquet specification, and its use is not encouraged.
- **ROW_GROUP_SIZE=<integer>:** Defaults to `65536`. Maximum number of rows per group.
- **GEOMETRY_NAME=value:** Defaults to `geometry`. Name of geometry column.
- **FID=value:** Name of the FID (Feature Identifier) column to create. If none is specified, no FID column is created. Note that if using ogr2ogr with the Parquet driver as the target driver and a source layer that has a named FID column, this FID column name will be automatically used to set the FID layer creation option of the Parquet driver (unless `-lco FID=` is used to set an empty name)
- **POLYGON_ORIENTATION=[COUNTERCLOCKWISE/UNMODIFIED]:** Defaults to `COUNTERCLOCKWISE`. Whether exterior rings of polygons should be counterclockwise oriented (and interior rings clockwise oriented), or left to their original orientation.
- **EDGES=[PLANAR/SPHERICAL]:** Defaults to `PLANAR`. How to interpret the edges of the geometries: whether the line between two points is a straight cartesian line (PLANAR) or the shortest line on the sphere (geodesic line) (SPHERICAL).
- **CREATOR=value:** Name of creating application.
- **WRITE_COVERING_BBOX=[AUTO/YES/NO]:** (GDAL >= 3.9) Defaults to `AUTO`. Whether to write xmin/ymin/xmax/ymax columns with the bounding box of geometries. Writing the geometry bounding box may help applications to perform faster spatial filtering. Writing a geometry bounding box is less necessary for the GeoArrow geometry encoding than for the default WKB, as implementations may be able to directly use the geometry columns. If the *USE_PARQUET_GEO_TYPES* layer creation option is set to `ONLY`, and *WRITE_COVERING_BBOX* is set or let to its default `AUTO` value, no covering bounding box columns is written.
- **COVERING_BBOX_NAME=<string>:** (GDAL >= 3.13) Name of the bounding box of geometries Only used if *WRITE_COVERING_BBOX* is set. If not set, it defaults to the geometry column name, suffixed with `_bbox`.
- **USE_PARQUET_GEO_TYPES=[YES/NO/ONLY]:** (GDAL >= 3.12) Defaults to `NO`. Only available with libarrow >= 21. Whether to use Parquet Geometry/Geography logical types (introduced in libarrow 21), when using the default GEOMETRY_ENCODING=WKB encoding.
  - `YES`: use the Geometry logical type (or the Geography one if the EDGES=SPHERICAL creation option is also set), and also write file-level GeoParquet metadata. Such files can be read by older GDAL, but require libarrow >= 20.
  - `NO` (default): only file-level GeoParquet metadata is written. Such files can be read by older GDAL and libarrow versions.
  - `ONLY`: use the Geometry logical type (or the Geography one if the EDGES=SPHERICAL creation option is also set), but do not write file-level GeoParquet metadata. Such files will only be fully compatible of GDAL

>= 3.12 and libarrow >= 21. With libarrow 20, the geometry column of such files will only be recognized if it is among one of the GEOM_POSSIBLE_NAMES open option value, and the CRS of such files will not be recognized. With older libarrow, such files cannot be opened at all.

- **SORT_BY_BBOX=[YES/NO]:** (GDAL >= 3.9) Defaults to `NO`. Whether features should be sorted based on the bounding box of their geometries, before being written in the final file. Sorting them enables faster spatial filtering on reading, by grouping together spatially close features in the same group of rows. Note however that enabling this option involves creating a temporary GeoPackage file (in the same directory as the final Parquet file), and thus requires temporary storage (possibly up to several times the size of the final Parquet file, depending on Parquet compression) and additional processing time.

  The efficiency of spatial filtering depends on the ROW_GROUP_SIZE. If it is too large, too many features that are not spatially close will be grouped together. If it is too small, the file size will increase, and extra processing time will be necessary to browse through the row groups.

  Note also that when this option is enabled, the Arrow writing API (which is for example triggered when using ogr2ogr to convert from Parquet to Parquet), fallbacks to the generic implementation, which does not support advanced Arrow types (lists, maps, etc.).

- **TIMESTAMP_WITH_OFFSET=[AUTO/YES/NO]:** (GDAL >= 3.13) Defaults to `AUTO`. Whether OGR datetime fields should be written as Arrow timestamp with offset fields, following the Timestamp With Offset extension specification. Such fields store both the datetime as a timestamp expressed in the UTC timezone and the offset to UTC of the timezone in which the datetime is defined. In AUTO mode, they are used as soon as a DateTime field reports a mixed time zone flag (i.e. *`OGRFieldDefn::GetTZFlag()`* returns `OGR_TZFLAG_MIXED_TZ`). As few drivers are able to automatically set this flag, it may be useful to override the flag by setting this option to YES. Setting it to NO forces the use of a DateTime field with the UTC timezone.

7.62.5 SQL support

SQL statements are run through the OGR SQL engine. Statistics can be used to speed-up evaluations of SQL requests like: "SELECT MIN(colname), MAX(colname), COUNT(colname) FROM layername"

7.62.6 Dataset/partitioning read support

Starting with GDAL 3.6.0, the driver can read directories that contain several Parquet files, and expose them as a single layer. This support is only enabled if the driver is built against the `arrowdataset` C++ library.

It is also possible to force opening single Parquet file in that mode by prefixing their filename with `PARQUET:`.

Optimized spatial and attribute filtering for Arrow datasets is available since GDAL 3.10.

7.62.7 Metadata

Added in version 3.9.0.

Layer metadata can be read and written. It is serialized as JSON content in a `gdal:metadata` domain.

7.62.8 Multithreading

Starting with GDAL 3.6.0, the driver will use up to 4 threads for reading (or the maximum number of available CPUs returned by *`CPLGetNumCPUs()`* if it is lower by 4). This number can be configured with the configuration option *`GDAL_NUM_THREADS`*, which can be set to an integer value or `ALL_CPUS`.

7.62.9 Update support

Added in version 3.12.0.

The driver supports adding, updating, removing features and fields. Note that this is accomplished by rewriting the whole file during *GDALDataset::FlushCache()* or when closing the dataset, and keeping into memory all modifications between two flushes.

If the file to be updated has been created by GDAL >= 3.12, the creation options to create the original file will be re-applied when creating the updated file. Stability of feature ids is only guaranteed if the original file has been created with an explicit `FID` creation option.

If the file to be updated has been created outside of GDAL, the original non-scalar/nested field types will not always be preserved, but a JSON representation may be used instead.

7.62.10 Validation script

The validate_geoparquet.py Python script can be used to check compliance of a Parquet file against the GeoParquet specification.

To validate only metadata:

```
python3 validate_geoparquet.py my_geo.parquet
```

To validate metadata and check content of geometry column(s):

```
python3 validate_geoparquet.py --check-data my_geo.parquet
```

7.62.11 Conda-forge package

The driver can be installed as a plugin for the `libgdal` conda-forge package with:

```
conda install -c conda-forge libgdal-arrow-parquet
```

7.62.11.1 Standalone plugin compilation

Added in version 3.10.

While this driver may be built as part of a whole GDAL build, either in libgdal itself, or as a plugin, it is also possible to only build this driver as a plugin, against an already built libgdal.

The version of the GDAL sources used to build the driver must match the version of the libgdal it is built against.

For example, from a "build_parquet" directory under the root of the GDAL source tree:

```
cmake -S ../ogr/ogrsf_frmts/parquet -DCMAKE_PREFIX_PATH=/path/to/GDAL_installation_
↪prefix -DArrow_DIR=/path/to/lib/cmake/Arrow -DParquet_DIR=/path/to/lib/cmake/Parquet
cmake --build .
```

Note that such a plugin, when used against a libgdal not aware of it, will be systematically loaded at GDAL driver initialization time, and will not benefit from *deferred plugin loading capabilities <rfc-96>*. For that, libgdal itself must be built with the CMake variable OGR_REGISTER_DRIVER_PARQUET_FOR_LATER_PLUGIN=ON set.

7.62.12 Links

- Apache Parquet home page
- Parquet file format

- GeoParquet specification
- Related driver: *Arrow driver*

7.63 PDF -- Geospatial PDF

Driver short name

PDF

Build dependencies

none for write support, Poppler/PoDoFo/PDFium for read support

Refer to the *PDF raster* documentation page for common documentation of the raster and vector sides of the driver.

7.63.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.63.2 Vector support

This driver can read and write geospatial PDF with vector features. Vector read support requires linking to one of the above mentioned dependent libraries, but write support does not. The driver can read vector features encoded according to PDF's logical structure facilities (as described by "§10.6 - Logical Structure" of PDF spec), or retrieve only vector geometries for other vector PDF files.

If there is no such logical structure, the driver will not try to interpret the vector content of the PDF, unless you defined the `OGR_PDF_READ_NON_STRUCTURED` configuration option to YES.

7.63.3 Feature style support

For write support, the driver has partial support for the style information attached to features, encoded according to the *Feature Style Specification*.

The following tools are recognized:

- For points, LABEL and SYMBOL.

- For lines, PEN.
- For polygons, PEN and BRUSH.

The supported attributes for each tool are summed up in the following table:

Tool	Supported attributes	Example
PEN	color (c); width (w); dash pattern (p)	PEN(c:#FF0000,w:5px)
BRUSH	foreground color (fc)	BRUSH(fc:#0000FF)
LABEL	text (t), limited to ASCII strings; font name (f), see note below; font size (s); bold (bo); italic (it); text color (c); x and y offsets (dx, dy); angle (a); anchor point (p), values 1 through 9; stretch (w)	LABEL(c:#000000,t:"Hello World!",s:5g)
SYMBOL	id (ogr-sym-0 to ogr-sym-9, and filenames for raster symbols); color (c); size (s)	SYMBOL(c:#00FF00,id:"ogr-sym-3",s:10) SYMBOL(c:#00000080,id:"a_symbol.png")

Alpha values are supported for colors to control the opacity. If not specified, for BRUSH, it is set at 50% opaque.

For SYMBOL with a bitmap name, only the alpha value of the color specified with 'c' is taken into account.

A font name starting with "Times" or containing the string "Serif" (case sensitive) will be treated as Times. A font name starting with "Courier" or containing the string "Mono" (case sensitive) will be treated as Courier. All other font names will be treated as Helvetica.

7.63.4 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The following configuration options are available:

- **OGR_PDF_READ_NON_STRUCTURED=[YES/NO]:** Defaults to `NO`. Attempt to read vector content even if no logical structure is present. See *Vector support*.

7.63.5 See Also

- *PDF raster* documentation page
- *Feature Style Specification*

7.64 PDS - Planetary Data Systems TABLE

Driver short name

PDS

Driver built-in by default

This driver is built-in by default

This driver reads TABLE objects from PDS datasets. Note there is a GDAL PDS driver to read the raster IMAGE objects from PDS datasets.

The driver must be provided with the product label file (even when the actual data is placed in a separate file).

If the label file contains a *TABLE* object, it will be read as the only layer of the dataset. If no *TABLE* object is found, the driver will look for all objects containing the TABLE string and read each one in a layer.

ASCII and BINARY tables are supported. The driver can retrieve the field descriptions from inline COLUMN objects or from a separate file pointed by ^STRUCTURE.

If the table has a LONGITUDE and LATITUDE columns of type REAL and with UNIT=DEGREE, they will be used to return POINT geometries.

7.64.1 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.64.2 See Also

- Description of PDS format (see Annex A.29 from StdRef_20090227_v3.8.pdf)

7.65 PostgreSQL SQL Dump

Driver short name

PGDump

Driver built-in by default

This driver is built-in by default

This write-only driver implements support for generating a SQL dump file that can later be injected into a live PostgreSQL instance. It supports PostgreSQL extended with the PostGIS geometries.

This driver is very similar to the PostGIS shp2pgsql utility.

Most creation options are shared with the regular PostgreSQL driver.

The PGDump driver supports creating tables with multiple PostGIS geometry columns (following rfc-41)

7.65.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.65.2 Creation options

7.65.2.1 Dataset Creation Options

Dataset creation options can be specified in command-line tools using the syntax `-dsco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following dataset creation options are supported:

- **LINEFORMAT=[CRLF/LF]:** By default files are created with the line termination conventions of the local platform (CR/LF on win32 or LF on all other systems). This may be overridden through use of the *LINEFORMAT* dataset creation option which may have a value of **CRLF** (DOS format) or **LF** (Unix format).

7.65.2.2 Layer Creation Options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALDatasetCreateLayer()* (C) or *Dataset.CreateLayer* (Python). The following layer creation options are supported:

- **GEOM_TYPE=[geometry/geography]:** Defaults to `geometry`. The *GEOM_TYPE* layer creation option can be set to one of "geometry" or "geography" (PostGIS >= 1.5) to force the type of geometry used for a table.
- **LAUNDER=[YES/NO]:** Defaults to `YES`. This may be "YES" to force new fields created on this layer to have their field names "laundered" into a form more compatible with PostgreSQL. This converts to lower case and converts some special characters like "-" and "#" to "_". If "NO" exact names are preserved. If enabled the table (layer) name will also be laundered.
- **LAUNDER_ASCII=[YES/NO]:** (GDAL >= 3.9) Defaults to `NO`. Implies LAUNDER=YES, with the extra substitution of UTF-8 accented characters in the Latin-1 Supplement and Latin Extented-A sets with the closest ASCII letter. Other non-ASCII characters are replaced with underscore. Consequently this option is not appropriate for non-Latin languages.
- **PRECISION=[YES/NO]:** Defaults to `YES`. This may be "YES" to force new fields created on this layer to try and represent the width and precision information, if available using NUMERIC(width,precision) or CHAR(width) types. If "NO" then the types FLOAT8, INTEGER and VARCHAR will be used instead. The default is "YES".
- **DIM=[2/3/XYM/XYZM]:** Control the dimension of the layer. Important to set to 2 for 2D layers with PostGIS 1.0+ as it has constraints on the geometry dimension during loading.
- **GEOMETRY_NAME=value:** Set name of geometry column in new table. If omitted it defaults to *wkb_geometry* for GEOM_TYPE=geometry, or *the_geog* for GEOM_TYPE=geography.

- **SCHEMA=value:** Set name of schema for new table. Using the same layer name in different schemas is supported, but not in the public schema and others.
- **CREATE_SCHEMA=[ON/OFF]:** To be used in combination with *SCHEMA*. Set to ON by default so that the CREATE SCHEMA instruction is emitted. Turn to OFF to prevent CREATE SCHEMA from being emitted.
- **SPATIAL_INDEX=[NONE/GIST/SPGIST/BRIN]:** Defaults to `GIST`. YES/NO for earlier versions and backward compatibility: Set to GIST by default. Creates a spatial index (GiST) on the geometry column to speed up queries (Has effect only when PostGIS is available). Set to NONE to disable. BRIN is only available with PostgreSQL >= 9.4 and PostGIS >= 2.3. SPGIST is only available with PostgreSQL >= 11 and PostGIS >= 2.5
- **GEOM_COLUMN_POSITION=[IMMEDIATE/END]:** (GDAL >= 3.7) Defaults to `IMMEDIATE`. If set to IMMEDIATE (the default), geometry columns are created in the table structure as soon as OGR is instructed to create them (which typically means that the default geometry column is created just after the FID column and before the non-spatial columns). If set to END, the geometry columns are created in the table structure after non-spatial columns.
- **TEMPORARY=[ON/OFF]:** Defaults to `OFF`. Creates a temporary table instead of a permanent one.
- **UNLOGGED=[ON/OFF]:** Defaults to `OFF`. Whether to create the table as a unlogged one. Unlogged tables are only supported since PostgreSQL 9.1, and GiST indexes used for spatial indexing since PostgreSQL 9.3.
- **WRITE_EWKT_GEOM=[ON/OFF]:** Defaults to `OFF`. Turn to ON to write EWKT geometries instead of HEX geometries. This option will have no effect *PG_USE_COPY* environment variable is to YES.
- **CREATE_TABLE=[ON/OFF]:** Defaults to `ON`. Set to ON by default so that tables are recreated if necessary. Turn to OFF to disable this and use existing table structure.
- **DROP_TABLE=[ON/OFF/IF_EXISTS]:** Defaults to `IF_EXISTS`. Set to ON so that tables are destroyed before being recreated. Set to OFF to prevent DROP TABLE from being emitted. Set to IF_EXISTS in order DROP TABLE IF EXISTS to be emitted (needs PostgreSQL >= 8.2)
- **SKIP_CONFLICTS=[ON/OFF]:** (GDAL >= 3.12) Defaults to `OFF`. Description Set to "ON" to ignore conflicts when inserting features (only applies when PG_USE_COPY is off).
- **SRID=value:** Set the SRID of the geometry. Defaults to -1, unless a SRS is associated with the layer. In the case, if the EPSG code is mentioned, it will be used as the SRID. (Note: the spatial_ref_sys table must be correctly populated with the specified SRID)
- **NONE_AS_UNKNOWN=[YES/NO]:** Can be set to YES to force non-spatial layers (wkbNone) to be created as spatial tables of type GEOMETRY (wkbUnknown). Defaults to NO, in which case a regular table is created and not recorded in the PostGIS geometry_columns table.
- **FID=value:** Defaults to `ogc_fid`. Name of the FID column to create. Starting with GDAL 3.7, if set explicitly to the empty string, disables the creation of a FID column.
- **FID64=[TRUE/FALSE]:** Defaults to `FALSE`. This may be "TRUE" to create a FID column that can support 64 bit identifiers.
- **EXTRACT_SCHEMA_FROM_LAYER_NAME=[YES/NO]:** Defaults to `YEs`. Can be set to NO to avoid considering the dot character as the separator between the schema and the table name.
- **COLUMN_TYPES=value:** A list of strings of format field_name=pg_field_type (separated by comma) that should be use when CreateField() is invoked on them. This will override the default choice that OGR would have made. This can for example be used to create a column of type HSTORE.
- **POSTGIS_VERSION=[1.5/2.0/2.2]:** Defaults to 2.2 starting with GDAL 3.2 (1.5 previously) PostGIS 2.0 encodes differently non-linear geometry types. And 2.2 brings special handling for POINT EMPTY geometries.
- **DESCRIPTION=value:** Description string to put in the pg_description system table. The description can also be written with SetMetadataItem("DESCRIPTION", description_string). Descriptions are preserved by default by ogr2ogr, unless the -nomd option is used.

7.65.2.3 Environment variables

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The following configuration options are available:

- **PG_USE_COPY=value:** This may be "YES" for using COPY for inserting data to Postgresql. COPY is significantly faster than INSERT. COPY is used by default when inserting from a table that has just been created.

  > **Warning**
  >
  > At time of writing, PgPoolII is not compatible with COPY mode as used by the OGR PostgreSQL driver. Thus you should force PG_USE_COPY=NO when using PgPoolII.

  > **Warning**
  >
  > It is not always possible to use COPY. In particular if the input layer has columns with a DEFAULT value specified and that the feature has no value set for that given field, the driver will default to INSERT even if instructed to use COPY via this option.

7.65.2.4 VSI Virtual File System API support

The driver supports writing to files managed by VSI Virtual File System API, which include "regular" files, as well as files in the /vsizip/, /vsigzip/ domains.

Writing to /dev/stdout or /vsistdout/ is also supported.

7.65.3 Examples

7.65.3.1 Example 1: Simple translation of a shapefile into PostgreSQL into a file abc.sql

The table 'abc' will be created with the features from abc.shp and attributes from abc.dbf. The SRID is specified. *`PG_USE_COPY`* is set to YES to improve the performance.

```
ogr2ogr --config PG_USE_COPY YES -f PGDump abc.sql abc.shp -lco SRID=32631
```

7.65.3.2 Example 2: Piping the output of the PGDump driver into the psql utility.

```
ogr2ogr --config PG_USE_COPY YES -f PGDump /vsistdout/ abc.shp | psql -d my_dbname -f -
```

7.65.3.3 See Also

- *OGR PostgreSQL driver Page*
- PostgreSQL Home Page
- PostGIS
- PostGIS / OGR Wiki Examples Page

7.66 ESRI Personal GeoDatabase

Driver short name

PGeo

Build dependencies

ODBC library

OGR optionally supports reading ESRI Personal GeoDatabase .mdb files via ODBC. Personal GeoDatabase is a Microsoft Access database with a set of tables defined by ESRI for holding geodatabase metadata, and with geometry for features held in a BLOB column in a custom format (essentially Shapefile geometry fragments). This driver accesses the personal geodatabase via ODBC but does not depend on any ESRI middle-ware.

Personal Geodatabases are accessed by passing the file name of the .mdb file to be accessed as the data source name.

In order to facilitate compatibility with different configurations, the *PGEO_DRIVER_TEMPLATE* config option was added to provide a way to programmatically set the DSN with the filename as an argument. In cases where the driver name is known, this allows for the construction of the DSN based on that information in a manner similar to the default (used for Windows access to the Microsoft Access Driver).

OGR treats all feature tables as layers. Most geometry types should be supported, including 3D data. Measure information (m value) is also supported. Coordinate system information should be properly associated with layers.

Currently the OGR Personal Geodatabase driver does not take advantage of spatial indexes for fast spatial queries, though that may be added in the future.

The Personal GeoDatabase format does not strictly differentiate between multi and single geometry types for polygon or line layers, and it is possible for a polygon or line layer to contain a mix of both single and multi type geometries. Accordingly, in order to provide predictable geometry types, the GDAL driver will always report the type of a line layer as wkbMultiLineString, and a polygon layer as wkbMultiPolygon. Single-part line or polygon features in the database will be promoted to multilinestrings or multipolygons during reading.

By default, SQL statements are passed directly to the MDB database engine. It's also possible to request the driver to handle SQL commands with *OGR SQL* engine, by passing **"OGRSQL"** string to the ExecuteSQL() method, as name of the SQL dialect.

7.66.1 Special SQL requests

"GetLayerDefinition a_layer_name" and "GetLayerMetadata a_layer_name" can be used as special SQL requests to get respectively the definition and metadata of a Personal GeoDatabase table as XML content.

7.66.2 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). The following configuration options are available:

- **PGEO_DRIVER_TEMPLATE=value:** Allows setting the DSN with the filename as an argument, e.g. `DRIVER=Microsoft Access Driver (*.mdb, *.accdb);DBQ=%s`.

7.66.3 Dataset open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are supported:

- **LIST_ALL_TABLES=[YES/NO]:** (GDAL >= 3.4) This may be "YES" to force all tables, including system and internal tables (such as the GDB_* tables) to be listed

7.66.4 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

7.66.5 Field domains

Added in version 3.4.

Coded and range field domains are supported.

7.66.6 Relationships

Added in version 3.6.

Relationship retrieval is supported.

7.66.7 How to use PGeo driver with unixODBC and MDB Tools (on Unix and Linux)

This article gives step-by-step explanation of how to use OGR with unixODBC package and how to access Personal Geodatabase with PGeo driver.

7.66.7.1 Prerequisites

1. Install unixODBC >= 2.2.11
2. Install MDB Tools. The official upstream of MDB Tools is maintained at https://github.com/mdbtools/mdbtools Version 0.9.4 or later is recommended for best compatibility with the PGeo driver. With the latest version no additional configuration (the following) is required on Debian Linux and `ogrinfo sample.mdb` works fine.

(On Ubuntu : sudo apt-get install unixodbc libmdbodbc)

(On Debian : sudo apt-get install unixodbc odbc-mdbtools)

7.66.7.2 Configuration

There are two configuration files for unixODBC:

- odbcinst.ini - this file contains definition of ODBC drivers available to all users; this file can be found in /etc directory or location given as --sysconfdir if you did build unixODBC yourself.
- odbc.ini - this file contains definition of ODBC data sources (DSN entries) available to all users.
- ~/.odbc.ini - this is the private file where users can put their own ODBC data sources.

Editing the odbc.ini files is only required if you want to setup an ODBC Data Source Name (DSN) so that Personal Geodatabase files can be directly accessed via DSN. This is entirely optional, as the PGeo driver will automatically handle the required connection parameters for you if a direct .mdb file name is used instead.

Format of configuration files is very simple:

```
[section_name]
entry1 = value
entry2 = value
```

For more details, refer to unixODBC manual.

1. ODBC driver configuration

First, you need to configure ODBC driver to access Microsoft Access databases with MDB Tools. Add following definition to your odbcinst.ini file.

```
[Microsoft Access Driver (*.mdb)]
Description = MDB Tools ODBC drivers
Driver      = /usr/lib/libmdbodbc.so.0
Setup       =
FileUsage   = 1
CPTimeout   =
CPReuse     =
```

- [Microsoft Access Driver (*.mdb)] - remember to use "Microsoft Access Driver (*.mdb)" as the name of section because PGeo driver composes ODBC connection string for Personal Geodatabase using "DRIVER=Microsoft Access Driver (*.mdb);" string.
- Description - put short description of this driver definition.
- Driver - full path of ODBC driver for MDB Tools.

2. ODBC data source configuration (optional)

In this section, I use 'sample.mdb' as a name of Personal Geodatabase, so replace this name with your own database.

Create .odbc.ini file in your HOME directory:

```
$ touch ~/.odbc.ini
```

Put following ODBC data source definition to your .odbc.ini file:

```
[sample_pgeo]
Description = Sample PGeo Database
Driver      = Microsoft Access Driver (*.mdb)
Database    = /home/mloskot/data/sample.mdb
Host        = localhost
Port        = 1360
User        = mloskot
Password    =
Trace       = Yes
TraceFile   = /home/mloskot/odbc.log
```

Step by step explanation of DSN entry:

- [sample_pgeo] - this is name of ODBC data source (DSN). You will refer to your Personal Geodatabase using this name. You can use your own name here.
- Description - short description of the DSN entry.
- Driver - full name of driver defined in step 1. above.
- Database - full path to .mdb file with your Personal Geodatabase.

- Host, Port, User and Password entries are not used by MDB Tools driver.

7.66.7.3 Testing PGeo driver with ogrinfo

Now, you can try to access PGeo data source with ogrinfo.

First, check if you have PGeo driver built in OGR:

```
$ ogrinfo --formats
Supported Formats:
  ESRI Shapefile
  ...
  PGeo
  ...
```

Now, you can access your Personal Geodatabase. If you've setup a DSN for the Personal Geodatabase (as detailed in section 2 above), the data source should be PGeo:<DSN> where <DSN> is the name of DSN entry you put to your .odbc.ini.

Alternatively, you can pass a .mdb filename directly to OGR to avoid manual creation of the DSN.

```
ogrinfo PGeo:sample_pgeo
INFO: Open of `PGeo:sample_pgeo'
using driver `PGeo' successful.
1. ...
```

After you run the command above, you should get list of layers stored in your geodatabase.

Now, you can try to query details of particular layer:

```
ogrinfo PGeo:sample_pgeo <layer name>
INFO: Open of `PGeo:sample_pgeo'
using driver `PGeo' successful.

Layer name: ...
```

7.66.8 Resources

- About ESRI Geodatabase
- MDB Tools project home

7.67 PostgreSQL / PostGIS

Driver short name

PostgreSQL

Build dependencies

PostgreSQL client library (libpq)

This driver implements support for access to spatial tables in PostgreSQL extended with the PostGIS spatial data support. Some support exists in the driver for use with PostgreSQL without PostGIS but with less functionalities.

This driver requires a connection to a Postgres database. If you want to prepare a SQL dump to inject it later into a Postgres database, you can instead use the *PostgreSQL SQL Dump driver*.

You can find additional information on the driver in the *Advanced OGR PostgreSQL driver Information* page.

Starting with GDAL 3.9, only PostgreSQL >= 9 and PostGIS >= 2 are supported.

7.67.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

7.67.2 Connecting to a database

To connect to a Postgres datasource, use a connection string specifying the database name, with additional parameters as necessary. The PG: prefix is used to mark the name as a postgres connection string.

```
PG:dbname=databasename
```

or

```
PG:"dbname='databasename' host='addr' port='5432' user='x' password='y'"
```

In this syntax each parameter setting is in the form keyword = value. Spaces around the equal sign are optional. To write an empty value, or a value containing spaces, surround it with single quotes, e.g., keyword = 'a value'. Single quotes and backslashes within the value must be escaped with a backslash, i.e., ' and \.

Starting with GDAL 3.1 also this syntax is supported:

```
PG:service=servicename
```

Starting with GDAL 3.4, the URI syntax is also supported

```
postgresql://[user[:password]@][netloc][:port][/dbname][?param1=value1&...]
```

It's also possible to omit the database name and connect to a *default* database, with the same name as the user name.
Note: We use PQconnectdb() to make the connection. See details from PostgreSQL libpq documentation).

7.67.3 Geometry columns

If the *geometry_columns* table exists (i.e. PostGIS is enabled for the accessed database), then all tables and named views listed in the *geometry_columns* table will be treated as OGR layers. Otherwise (PostGIS disabled for the accessed database), all regular user tables and named views will be treated as layers.

The driver also supports the geography column type.

The driver also supports reading and writing the following non-linear geometry types :CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE and MULTISURFACE

7.67.4 SQL statements

The PostgreSQL driver passes SQL statements directly to PostgreSQL by default, rather than evaluating them internally when using the ExecuteSQL() call on the OGRDataSource, or the -sql command option to ogr2ogr. Attribute query expressions are also passed directly through to PostgreSQL. It's also possible to request the ogr Pg driver to handle SQL commands with the *OGR SQL* engine, by passing **"OGRSQL"** string to the ExecuteSQL() method, as the name of the SQL dialect.

Note that the PG driver uses PostgreSQL cursors to browse through the result set of a ExecuteSQL() request, and that, at time of writing, PostgreSQL default settings aren't optimized when the result set is small enough to fit in one result page. If you experiment bad performance, specifying the `PRELUDE_STATEMENTS=SET cursor_tuple_fraction = 1.0;` open option might help.

The PostgreSQL driver in OGR supports the OGRDataSource::StartTransaction(), OGRDataSource::CommitTransaction() and OGRDataSource::RollbackTransaction() calls in the normal SQL sense.

7.67.5 Creation Issues

The PostgreSQL driver does not support creation of new datasets (a database within PostgreSQL), but it does allow creation of new layers within an existing database.

As mentioned above the type system is impoverished, and many OGR types are not appropriately mapped into PostgreSQL.

If the database has PostGIS types loaded (i.e. the geometry type), newly created layers will be created with the PostGIS Geometry type. Otherwise they will use OID.

By default it is assumed that text being sent to Postgres is in the UTF-8 encoding. This is fine for plain ASCII, but can result in errors for extended characters (ASCII 155+, LATIN1, etc). While OGR provides no direct control over this, you can set the PGCLIENTENCODING environment variable to indicate the format being provided. For instance, if your text is LATIN1 you could set the environment variable to LATIN1 before using OGR and input would be assumed to be LATIN1 instead of UTF-8. An alternate way of setting the client encoding is to issue the following SQL command with ExecuteSQL() : "SET client_encoding TO encoding_name" where encoding_name is LATIN1, etc. Errors can be caught by enclosing this command with a CPLPushErrorHandler()/CPLPopErrorHandler() pair.

7.67.6 Updating existing tables

When data is appended to an existing table (for example, using the `-append` option in `ogr2ogr`) the driver will, by default, emit an INSERT statement for each row of data to be added. This may be significantly slower than the COPY-based approach taken when creating a new table, but ensures consistency of unique identifiers if multiple connections are accessing the table simultaneously.

If only one connection is accessing the table when data is appended, the COPY-based approach can be chosen by setting the config option `PG_USE_COPY` to `YES`, which may significantly speed up the operation.

7.67.6.1 Dataset open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are supported:

- **DBNAME=<string>:** Database name.
- **PORT=<integer>:** Port.
- **USER=<string>:** User name.
- **PASSWORD=<string>:** Password.
- **HOST=<string>:** Server hostname.
- **SERVICE=<string>:** (GDAL >= 3.1) Service name
- **ACTIVE_SCHEMA=<string>:** Active schema.
- **SCHEMAS=value:** Restricted sets of schemas to explore (comma separated).
- **TABLES=value:** Restricted set of tables to list (comma separated). Each table name can be qualified by its schema, like `schema_name.table_name`. For tables with multiple geometry columns, the `table_name(geometry_column)` syntax can be used. Starting with GDAL 3.11, if the table name itself contains an open parenthesis `(`, or a backslash character `\`, they must be escaped with a preceding backslash character. e.g. `table_name \(with parenthesis)`. If that table name contains a comma, the table name must be quoted with double quotes. e.g. `first_table,"second table, with comma"`.
- **LIST_ALL_TABLES=[YES/NO]:** This may be "YES" to force all tables, including non-spatial ones, to be listed.
- **SKIP_VIEWS=[YES/NO]:** (GDAL >= 3.7) This may be "YES" to prevent views from being listed.
- **PRELUDE_STATEMENTS=value:** SQL statement(s) to send on the PostgreSQL client connection before any other ones. In case of several statements, they must be separated with the semi-column (;) sign. The driver will specifically recognize BEGIN as the first statement to avoid emitting BEGIN/COMMIT itself. This option may be useful when using the driver with pg_bouncer in transaction pooling, e.g. 'BEGIN; SET LOCAL statement_timeout TO "1h";'
- **CLOSING_STATEMENTS=value:** SQL statement(s) to send on the PostgreSQL client connection after any other ones. In case of several statements, they must be separated with the semi-column (;) sign. With the above example value for :oo::*PRELUDE_STATEMENTS*, the appropriate CLOSING_STATEMENTS would be "COMMIT".
- **SPATIAL_FILTER_INTERSECTION=[LOCAL/SERVER]:** (GDAL >= 3.13) Defaults to `LOCAL`. Since GDAL 3.13, spatial filters are evaluated using full geometry intersection rather than only bounding box intersection, as in earlier versions. The `SPATIAL_FILTER_INTERSECTION` open option controls where this intersection operation is performed. Even when the intersection is evaluated locally (which is the default), the server still applies a bounding box-based spatial filter to efficiently retrieve candidate features. Note that for PostGIS geometry columns of type GEOGRAPHY, the current implementation deals with geographies as if they were cartesian geometries.

7.67.6.2 Dataset Creation Options

None

7.67.6.3 Layer Creation Options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALDatasetCreateLayer()* (C) or *Dataset.CreateLayer* (Python). The following layer creation options are supported:

- **GEOM_TYPE=[geometry/geography/BYTEA/OID]:** The GEOM_TYPE layer creation option can be set to one of "geometry", "geography", "BYTEA" or "OID" to force the type of geometry used for a table. For a PostGIS database, "geometry" is the default value. PostGIS "geography" assumes a geographic SRS (before PostGIS 2.2, it was even required to be EPSG:4326), but the driver has no built-in reprojection logic so it is safest to use always *-t_srs EPSG:4326* (or the canonical geographic CRS corresponding to the projected CRS of interest to avoid datum transformations) with `GEOM_TYPE=geography`.
- **OVERWRITE=[YES/NO]:** This may be "YES" to force an existing layer of the desired name to be destroyed before creating the requested layer.
- **LAUNDER=[YES/NO]:** Defaults to `YES`. This may be "YES" to force new fields created on this layer to have their field names "laundered" into a form more compatible with PostgreSQL. This converts to lower case and converts some special characters like "-" and "#" to "_". If "NO" exact names are preserved. If enabled the table (layer) name will also be laundered.
- **LAUNDER_ASCII=[YES/NO]:** (GDAL >= 3.9) Defaults to `NO`. Implies LAUNDER=YES, with the extra substitution of UTF-8 accented characters in the Latin-1 Supplement and Latin Extented-A sets with the closest ASCII letter. Other non-ASCII characters are replaced with underscore. Consequently this option is not appropriate for non-Latin languages.
- **PRECISION=[YES/NO]:** Defaults to `YES`. This may be "YES" to force new fields created on this layer to try and represent the width and precision information, if available using NUMERIC(width,precision) or CHAR(width) types. If "NO" then the types FLOAT8, INTEGER and VARCHAR will be used instead.
- **DIM=[2/3/XYM/XYZM]:** Control the dimension of the layer. Important to set to 2 for 2D layers as it has constraints on the geometry dimension during loading.
- **GEOMETRY_NAME=value:** Set name of geometry column in new table. If omitted it defaults to *wkb_geometry* for GEOM_TYPE=geometry, or *the_geog* for GEOM_TYPE=geography.
- **SCHEMA=value:** Set name of schema for new table. Using the same layer name in different schemas is supported, but not in the public schema and others. Note that using the -overwrite option of ogr2ogr and -lco SCHEMA= option at the same time will not work, as the ogr2ogr utility will not understand that the existing layer must be destroyed in the specified schema. Use the -nln option of ogr2ogr instead, or better the active_schema connection string. See below example.
- **SPATIAL_INDEX=[NONE/GIST/SPGIST/BRIN]:** Defaults to `GIST`. Creates a spatial index (GiST) on the geometry column to speed up queries (Has effect only when PostGIS is available). Set to NONE to disable. BRIN is only available with PostgreSQL >= 9.4 and PostGIS >= 2.3. SPGIST is only available with PostgreSQL >= 11 and PostGIS >= 2.5
- **TEMPORARY=[ON/OFF]:** Defaults to `OFF`. Set to OFF by default. Creates a temporary table instead of a permanent one.
- **UNLOGGED=[ON/OFF]:** Set to OFF by default. Whether to create the table as a unlogged one. Unlogged tables are only supported since PostgreSQL 9.1, and GiST indexes used for spatial indexing since PostgreSQL 9.3.
- **NONE_AS_UNKNOWN=[YES/NO]:** Can bet set to YES to force non-spatial layers (wkbNone) to be created as spatial tables of type GEOMETRY (wkbUnknown). Defaults to NO, in which case a regular table is created and not recorded in the PostGIS geometry_columns table.
- **FID=value:** Defaults to `ogc_fid`. Name of the FID column to create.
- **FID64=[TRUE/FALSE]:** Defaults to `FALSE`. This may be "TRUE" to create a FID column that can support 64 bit identifiers.
- **EXTRACT_SCHEMA_FROM_LAYER_NAME=[YES/NO]:** Defaults to `YES`. Can be set to NO to avoid considering the dot character as the separator between the schema and the table name.

- **COLUMN_TYPES=value:** A list of strings of format field_name=pg_field_type (separated by comma) that should be use when CreateField() is invoked on them. This will override the default choice that OGR would have made. This can for example be used to create a column of type HSTORE.
- **DESCRIPTION=value:** Description string to put in the pg_description system table. On reading, if such a description is found, it is exposed in the DESCRIPTION metadata item. The description can also be written with SetMetadataItem("DESCRIPTION", description_string). Descriptions are preserved by default by ogr2ogr, unless the -nomd option is used.

7.67.6.4 Configuration Options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The following configuration options are available:

- **PG_USE_COPY=value:** This may be "YES" for using COPY for inserting data to Postgresql. COPY is significantly faster than INSERT. COPY is used by default when inserting from a table that has just been created.

> **Warning**
>
> At time of writing, PgPoolII is not compatible with COPY mode as used by the OGR PostgreSQL driver. Thus you should force PG_USE_COPY=NO when using PgPoolII.

> **Warning**
>
> It is not always possible to use COPY. In particular if the input layer has columns with a DEFAULT value specified and that the feature has no value set for that given field, the driver will default to INSERT even if instructed to use COPY via this option.

- **PGSQL_OGR_FID=value:** Set name of primary key instead of 'ogc_fid'. Only used when opening a layer whose primary key cannot be autodetected. Ignored by CreateLayer() that uses the FID creation option.
- **PG_LIST_ALL_TABLES=[YES/NO]:** Equivalent of *`LIST_ALL_TABLES`*.
- **PG_USE_BASE64=[YES/NO]:** Defaults to `NO`. If set to "YES", geometries will be fetched as BASE64 encoded EWKB instead of canonical HEX encoded EWKB. This reduces the amount of data to be transferred from 2 N to 1.333 N, where N is the size of EWKB data. However, it might be a bit slower than fetching in canonical form when the client and the server are on the same machine, so the default is NO.
- **OGR_PG_CURSOR_PAGE=value:** Set the cursor page size, or number of features that are fetched from the database and held in memory at a single time.
- **OGR_PG_RETRIEVE_FID=[YES/NO]:** Defaults to `YES`. If set to "YES" (the default), writing an OGRFeature will cause its FID to be set to the value assigned by the database. If a single feature is to be inserted multiple times, this option can be set to "NO" to allow the database to assign a new FID for each insertion.
- **OGR_TRUNCATE=value:** If set to "YES", the content of the table will be first erased with the SQL TRUNCATE command before inserting the first feature. This is an alternative to using the -overwrite flag of ogr2ogr, that avoids views based on the table to be destroyed. Typical use case: `ogr2ogr -append PG:dbname=foo abc.shp --config OGR_TRUNCATE YES`.
- **OGR_PG_ENABLE_METADATA=[YES/NO]:** (GDAL >= 3.9) Defaults to `YES`. If set to "YES" (the default), the driver will try to use the `ogr_system_tables.metadata` table to retrieve and store layer metadata. To be able to store metadata, the schema `ogr_system_tables` and the event trigger `ogr_system_tables_event_trigger_for_metadata` must already exist in the database, or the current user

must have sufficient rights (super-user rights for the event trigger) to be able to create them. If not, a warning will be raised.

- **OGR_PG_SKIP_CONFLICTS=[YES/NO]:** (GDAL >= 3.10) Defaults to `NO`. If set to "YES" (not the default), conflicts when inserting features will be skipped (requires OGR_PG_RETRIEVE_FID to be off and only applies when PG_USE_COPY is off).

7.67.6.5 Examples

Example 1: Simple translation of a shapefile into PostgreSQL.

The table 'abc' will be created with the features from abc.shp and attributes from abc.dbf. The database instance (warmerda) must already exist, and the table abc must not already exist.

```
ogr2ogr -f PostgreSQL PG:dbname=warmerda abc.shp
```

Example 2: Export a single Postgres table to GeoPackage:

```
ogr2ogr \
  -f GPKG output.gpkg \
  PG:"dbname='my_database'" "my_table"
```

Example 3: Export many Postgres tables to GeoPackage:

```
ogr2ogr \
  -f GPKG output.gpkg \
  PG:"dbname='my_database' tables='table_1,table_3'"
```

Example 4: Export a whole Postgres database to GeoPackage:

```
ogr2ogr \
  -f GPKG output.gpkg \
  PG:dbname=my_database
```

Example 5: Load a single layer GeoPackage into Postgres:

```
ogr2ogr \
  -f "PostgreSQL" PG:"dbname='my_database'" \
  input.gpkg \
  -nln "name_of_new_table"
```

Example 6: Writing multiple datasets to a single table

In this example we merge tiger line data from two different directories of tiger files into one table. Note that the second invocation uses -append and no *OVERWRITE=yes*.

```
ogr2ogr -f PostgreSQL PG:dbname=warmerda tiger_michigan \
     -lco OVERWRITE=yes CompleteChain
ogr2ogr -update -append -f PostgreSQL PG:dbname=warmerda tiger_ohio \
     CompleteChain
```

Example 7: Evaluating an SQL query

This example shows using ogrinfo to evaluate an SQL query statement within PostgreSQL. More sophisticated PostGIS specific queries may also be used via the -sql commandline switch to ogrinfo.

```
ogrinfo -ro PG:dbname=warmerda -sql "SELECT pop_1994 from canada where province_name =
↪'Alberta'"
```

Example 8: Listing layers on a different host

This example shows using ogrinfo to list PostgreSQL/PostGIS layers on a different host.

```
ogrinfo -ro PG:"host='myserver.velocet.ca' user='postgres' dbname='warmerda'"
```

Example 9: Use of PRELUDE_STATEMENTS and CLOSING_STATEMENTS

This example shows use of *PRELUDE_STATEMENTS* and *CLOSING_STATEMENTS* as destination open options of ogr2ogr!

```
ogrinfo PG:"dbname='mydb'" poly -doo "PRELUDE_STATEMENTS=BEGIN; SET LOCAL statement_
↪timeout TO '1h';" -doo CLOSING_STATEMENTS=COMMIT
```

7.67.7 FAQs

- **Why can't I see my tables? PostGIS is installed and I have data** You must have permissions on all tables you want to read *and* geometry_columns and spatial_ref_sys. Misleading behavior may follow without an error message if you do not have permissions to these tables. Permission issues on geometry_columns and/or spatial_ref_sys tables can be generally confirmed if you can see the tables by setting the configuration option *PG_LIST_ALL_TABLES* to YES. (e.g. `ogrinfo --config PG_LIST_ALL_TABLES YES PG:xxxxx`)

7.67.8 See Also

- *Advanced OGR PostgreSQL driver Information*
- *OGR PostgreSQL SQL Dump driver Page*
- PostgreSQL Home Page
- PostGIS
- PostGIS / OGR Wiki Examples Page

7.67.8.1 PostgreSQL / PostGIS - Advanced Driver Information

The information collected in that page deal with advanced topics, not found in the *OGR PostgreSQL driver Information* page.

Connection options related to schemas and tables

The database opening should be significantly faster than in previous versions, so using tables= or schemas= options will not bring further noticeable speed-ups.

The set of tables to be scanned can be overridden by specifying *tables=[schema.]table[(geom_column_name)][,[schema2.]table2[(geom_*
within the connection string. If the parameter is found, the driver skips enumeration of the tables as described in the next paragraph.

It is possible to restrict the schemas that will be scanned while establishing the list of tables. This can be done by specifying *schemas=schema_name[,schema_name2]* within the connection string. This can also be a way of speeding

up the connection to a PostgreSQL database if there are a lot of schemas. Note that if only one schema is listed, it will also be made automatically the active schema (and the schema name will not prefix the layer name). Otherwise, the active schema is still 'public', unless otherwise specified by the *active_schema=* option.

The active schema ('public' being the default) can be overridden by specifying *active_schema=schema_name* within the connection string. The active schema is the schema where tables are created or looked for when their name is not explicitly prefixed by a schema name. Note that this does not restrict the tables that will be listed (see *schemas=* option above). When getting the list of tables, the name of the tables within that active schema will not be prefixed by the schema name. For example, if you have a table 'foo' within the public schema, and a table 'foo' within the 'bar_schema' schema, and that you specify active_schema=bar_schema, 2 layers will be listed : 'foo' (implicitly within 'bar_schema') and 'public.foo'.

Multiple geometry columns

The PostgreSQL driver supports accessing tables with multiple PostGIS geometry columns.

OGR supports reading, updating, creating tables with multiple PostGIS geometry columns (following rfc-41) For such a table, a single OGR layer will be reported with as many geometry fields as there are geometry columns in the table.

For backward compatibility, it is also possible to query a layer with GetLayerByName() with a name formatted like 'foo(bar)' where 'foo' is a table and 'bar' a geometry column.

Layers

Even when PostGIS is enabled, if the user defines the environment variable `PG_LIST_ALL_TABLES=YES` (and does not specify tables=), all regular user tables and named views will be treated as layers. However, tables with multiple geometry column will only be reported once in that mode. So this variable is mainly useful when PostGIS is enabled to find out tables with no spatial data, or views without an entry in *geometry_columns* table.

In any case, all user tables can be queried explicitly with GetLayerByName()

Regular (non-spatial) tables can be accessed, and will return features with attributes, but not geometry. If the table has a "wkb_geometry" field, it will be treated as a spatial table. The type of the field is inspected to determine how to read it. It can be a PostGIS **geometry** field, which is assumed to come back in OGC WKT, or type BYTEA or OID in which case it is used as a source of OGC WKB geometry.

Tables inherited from spatial tables are supported.

If there is an "ogc_fid" field, it will be used to set the feature id of the features, and not treated as a regular field.

The layer name may be of the form "schema.table". The schema must exist, and the user needs to have write permissions for the target and the public schema.

If the user defines the environment variable

```
PG_SKIP_VIEWS=YES
```

(and does not specify tables=), only the regular user tables will be treated as layers. The default action is to include the views. This variable is particularly useful when you want to copy the data into another format while avoiding the redundant data from the views.

Named views

When PostGIS is enabled for the accessed database, named views are supported, provided that there is an entry in the *geometry_columns* tables. But, note that the AddGeometryColumn() SQL function doesn't accept adding an entry for a view (only for regular tables). So, that must usually be done by hand with a SQL statement like :

```
"INSERT INTO geometry_columns VALUES ( '', 'public', 'name_of_my_view', 'name_of_
↪geometry_column', 2, 4326, 'POINT');"
```

It is also possible to use named views without inserting a row in the geometry_columns table. For that, you need to explicitly specify the name of the view in the "tables=" option of the connection string. See above. The drawback is that OGR will not be able to report a valid SRS and figure out the right geometry type.

Retrieving FID of newly inserted feature

The FID of a feature (i.e. usually the value of the OGC_FID column for the feature) inserted into a table with CreateFeature(), in non-copy mode, will be retrieved from the database and can be obtained with GetFID(). One side-effect of this new behavior is that you must be careful if you reuse the same feature object in a loop that makes insertions. After the first iteration, the FID will be set to a non-null value, so at the second iteration, CreateFeature() will try to insert the new feature with the FID of the previous feature, which will fail as you cannot insert 2 features with same FID. So in that case you must explicitly reset the FID before calling CreateFeature(), or use a fresh feature object.

Snippet example in Python :

```
feat = ogr.Feature(lyr.GetLayerDefn())
for i in range(100):
    feat.SetFID(-1)  # Reset FID to null value
    lyr.CreateFeature(feat)
    print('The feature has been assigned FID %d' % feat.GetFID())
```

or :

```
for i in range(100):
    feat = ogr.Feature(lyr.GetLayerDefn())
    lyr.CreateFeature(feat)
    print('The feature has been assigned FID %d' % feat.GetFID())
```

Old GDAL behavior can be obtained by setting the configuration option *OGR_PG_RETRIEVE_FID* to FALSE.

Issues with transactions

Efficient sequential reading in PostgreSQL requires to be done within a transaction (technically this is a CURSOR WITHOUT HOLD). So the PG driver will implicitly open such a transaction if none is currently opened as soon as a feature is retrieved. This transaction will be released if ResetReading() is called (provided that no other layer is still being read).

If within such an implicit transaction, an explicit dataset level StartTransaction() is issued, the PG driver will use a SAVEPOINT to emulate properly the transaction behavior while making the active cursor on the read layer still opened.

If an explicit transaction is opened with dataset level StartTransaction() before reading a layer, this transaction will be used for the cursor that iterates over the layer. When explicitly committing or rolling back the transaction, the cursor will become invalid, and ResetReading() should be issued again to restart reading from the beginning.

As calling SetAttributeFilter() or SetSpatialFilter() implies an implicit ResetReading(), they have the same effect as ResetReading(). That is to say, while an implicit transaction is in progress, the transaction will be committed (if no other layer is being read), and a new one will be started again at the next GetNextFeature() call. On the contrary, if they are called within an explicit transaction, the transaction is maintained.

With the above rules, the below examples show the SQL instructions that are run when using the OGR API in different scenarios.

```
lyr1->GetNextFeature()              BEGIN (implicit)
                                    DECLARE cur1 CURSOR FOR SELECT * FROM lyr1
                                    FETCH 1 IN cur1
```

(continues on next page)

(continued from previous page)

```
lyr1->SetAttributeFilter('xxx')
     --> lyr1->ResetReading()      CLOSE cur1
                                   COMMIT (implicit)

lyr1->GetNextFeature()             BEGIN (implicit)
                                   DECLARE cur1 CURSOR  FOR SELECT * FROM lyr1 WHERE xxx
                                   FETCH 1 IN cur1

lyr2->GetNextFeature()             DECLARE cur2 CURSOR  FOR SELECT * FROM lyr2
                                   FETCH 1 IN cur2

lyr1->GetNextFeature()             FETCH 1 IN cur1

lyr2->GetNextFeature()             FETCH 1 IN cur2

lyr1->CreateFeature(f)             INSERT INTO cur1 ...

lyr1->SetAttributeFilter('xxx')
     --> lyr1->ResetReading()      CLOSE cur1
                                   COMMIT (implicit)

lyr1->GetNextFeature()             DECLARE cur1 CURSOR  FOR SELECT * FROM lyr1 WHERE xxx
                                   FETCH 1 IN cur1

lyr1->ResetReading()               CLOSE cur1

lyr2->ResetReading()               CLOSE cur2
                                   COMMIT (implicit)

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

ds->StartTransaction()             BEGIN

lyr1->GetNextFeature()             DECLARE cur1 CURSOR FOR SELECT * FROM lyr1
                                   FETCH 1 IN cur1

lyr2->GetNextFeature()             DECLARE cur2 CURSOR FOR SELECT * FROM lyr2
                                   FETCH 1 IN cur2

lyr1->CreateFeature(f)             INSERT INTO cur1 ...

lyr1->SetAttributeFilter('xxx')
     --> lyr1->ResetReading()      CLOSE cur1
                                   COMMIT (implicit)

lyr1->GetNextFeature()             DECLARE cur1 CURSOR  FOR SELECT * FROM lyr1 WHERE xxx
                                   FETCH 1 IN cur1

lyr1->ResetReading()               CLOSE cur1

lyr2->ResetReading()               CLOSE cur2
```

(continues on next page)

(continued from previous page)

```
ds->CommitTransaction()            COMMIT

~~~
```

```
ogrinfo -ro PG:'dbname=warmerda schemas=apt200810'
```

- This example shows how to convert a set of shapefiles inside the apt200810 directory into an existing Postgres schema apt200810. In that example, we could have use the schemas= option instead.

```
ogr2ogr -f PostgreSQL "PG:dbname=warmerda active_schema=apt200810" apt200810
```

- This example shows how to convert all the tables inside the schema apt200810 as a set of shapefiles inside the apt200810 directory. Note that the layer names will not be prefixed by apt200810 as only one schema is listed

```
ogr2ogr apt200810 PG:'dbname=warmerda schemas=apt200810'
```

- This example shows how to overwrite an existing table in an existing schema. Note the use of -nln to specify the qualified layer name.

```
ogr2ogr -overwrite -f PostgreSQL "PG:dbname=warmerda" mytable.shp mytable -nln␣
↪myschema.mytable
```

  Note that using -lco SCHEMA=mytable instead of -nln would not have worked in that case (see #2821 for more details).

  If you need to overwrite many tables located in a schema at once, the -nln option is not the more appropriate, so it might be more convenient to use the active_schema connection string. The following example will overwrite, if necessary, all the PostgreSQL tables corresponding to a set of shapefiles inside the apt200810 directory :

```
ogr2ogr -overwrite -f PostgreSQL "PG:dbname=warmerda active_schema=apt200810"␣
↪apt200810
```

See Also

- *OGR PostgreSQL driver Information*

7.68 PLScenes (Planet Labs Scenes/Catalog API)

Driver short name

PLScenes

Build dependencies

libcurl

This driver can connect to Planet Labs Data V1 API. GDAL/OGR must be built with Curl support in order for the PLScenes driver to be compiled.

Please consult the dedicated pages for each version of the API:

7.68.1 PLScenes (Planet Labs Scenes), Data V1 API

The driver supports read-only operations to list scenes and their metadata as a vector layer per item-types: "PSOrthoTile", "REOrthoTile", "PSScene3Band", "PSScene4Band", "REScene", "Landsat8L1G", "Sentinel2L1C". It can also access raster scenes.

7.68.1.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

7.68.1.2 Dataset name syntax

The minimal syntax to open a datasource is :

```
PLScenes:[options]
```

Additional optional parameters can be specified after the ':' sign. Currently the following one is supported :

- **version**=data_v1: To specify the API version to request.
- **api_key**=value: To specify the Planet API KEY. It is mandatory, unless it is supplied through the open option API_KEY, or the configuration option *PL_API_KEY*.
- **follow_links**=YES/NO: Whether assets links should be followed for each scene (vector). Getting assets links require a HTTP request per scene, which might be costly when enumerating through a lot of products. Defaults to NO.
- **scene**=scene_id: To specify the scene ID, when accessing raster data. Optional for vector layer access.
- **itemtypes**=name: To specify the item types name. Optional for vector layer access. Mandatory for raster access.
- **asset**=value: To specify the asset type (for raster fetching). Default is "visual". Optional for vector layer access. If the option is not specified and the 'visual' asset category does not exist for the scene (or if the value is set to 'list'), the returned dataset will have subdatasets for the available asset categories.
- **medata**=YES/NO: (Raster only) Whether scene metadata should be fetch from the API and attached to the raster dataset. Defaults to YES.

If several parameters are specified, they must be separated by a comma.

7.68.1.3 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are available:

- **VERSION=value:** Defaults to `DATA_V1`. To specify the API version to request.
- **API_KEY=value:** To specify the Planet API KEY.
- **FOLLOW_LINKS=[YES/NO]:** Defaults to `NO`. Whether assets links should be followed for each scene (vector). Getting assets links require a HTTP request per scene, which might be costly when enumerating through a lot of products.
- **SCENE=<scene_id>:** To specify the scene ID, when accessing raster data. Optional for vector layer access.
- **ITEMTYPES=<name>:** To specify the item types name. Optional for vector layer access. Mandatory for raster access.

- **ASSET=value:** Defaults to `visual`. To specify the asset type (for raster fetching). Optional for vector layer access. If the option is not specified and the 'visual' asset category does not exist for the scene (or if the value is set to 'list'), the returned dataset will have subdatasets for the available asset categories.
- **RANDOM_ACCESS=[YES/NO]:** Defaults to `YES`. Whether raster should be accessed in random access mode (but with potentially not optimal throughput). If NO, in-memory ingestion is done.
- **ACTIVATION_TIMEOUT=seconds:** Defaults to `3600`. Number of seconds during which to wait for asset activation (raster).
- **METADATA=[YES/NO]:** Defaults to `YES`. (Raster only) Whether scene metadata should be fetched from the API and attached to the raster dataset.

7.68.1.4 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The following configuration options are available:

- **PL_API_KEY=value:** To specify the Planet API key.
- **PLSCENES_PAGE_SIZE=value:** Defaults to `250`. Chunk size for fetching features. Maximum value is 250.

7.68.1.5 Attributes

The layer field definition is built from the "plscenesconf.json" file in the GDAL configuration. The links to downloadable products are in *asset_XXXXX_location* attributes where XXXXX is the asset category id, when they are active. Otherwise they should be activated by sending a POST request to the URL in the *asset_XXXXX_activate_link* attribute (what the raster driver does automatically)

Geometry

The footprint of each scene is reported as a MultiPolygon with a longitude/latitude WGS84 coordinate system (EPSG:4326).

Filtering

The driver will forward any spatial filter set with SetSpatialFilter() to the server. It also makes the same for simple attribute filters set with SetAttributeFilter(). Note that not all attributes support all comparison operators. Refer to comparator column in Metadata properties

Paging

Features are retrieved from the server by chunks of 250 by default (and this is the maximum value accepted by the server). This number can be altered with the *`PLSCENES_PAGE_SIZE`* configuration option.

Vector layer (scene metadata) examples

Example 1: Listing all scenes available (with the rights of the account)

```
ogrinfo -ro -al "PLScenes:" -oo API_KEY=some_value
```

or

```
ogrinfo -ro -al "PLScenes:api_key=some_value"
```

or

```
ogrinfo -ro -al "PLScenes:" --config PL_API_KEY some_value
```

Example 2: Listing all scenes available on PSOrthoTile item types, under a point of (lat,lon)=(40,-100)

```
ogrinfo -ro -al "PLScenes:" -oo API_KEY=some_value PSOrthoTile -spat -100 40 -100 40
```

Example 3: Listing all scenes available within a bounding box (lat,lon)=(40,-100) to (lat,lon)=(39,-99)

```
ogrinfo -ro -al "PLScenes:" -oo API_KEY=some_value -spat -100 40 -99 39
```

Example 4: Listing all scenes available matching criteria

```
ogrinfo -ro -al "PLScenes:" -oo API_KEY=some_value PSOrthoTile -where "acquired >= '2015/
↪03/26 00:00:00' AND cloud_cover < 10"
```

Example 5: Listing all downloadable scenes

```
ogrinfo -ro -al -q "PLScenes:" -oo API_KEY=some_value PSOrthoTile -where "permissions=
↪'assets:download'"
```

7.68.1.6 Raster access

Scenes can be accessed as raster datasets, provided that the scene ID is specified with the 'scene' parameter / *SCENE* open option. The 'itemtypes' parameter / *ITEMTYPES* open option must also be specified. The asset type (visual, analytic, ...) can be specified with the 'asset' parameter / *ASSET* open option. The scene id is the content of the value of the 'id' field of the features.

If the product is not already generated on the server, it will be activated, and the driver will wait for it to be available. The length of this retry can be configured with the *ACTIVATION_TIMEOUT* open option.

Raster access examples

Example 6: Displaying raster metadata

```
gdalinfo "PLScenes:scene=scene_id,itemtypes=itemypes,asset=analytic" -oo API_KEY=some_
↪value
```

or

```
gdalinfo "PLScenes:" -oo API_KEY=some_value -oo ITEMTYPES=itemtypes -oo SCENE=scene_id -
↪oo ASSET=analytic
```

Example 7: Converting/downloading a whole file

```
gdal_translate "PLScenes:" -oo API_KEY=some_value -oo SCENE=scene_id \
                -oo ITEMTYPES=itemtypes -oo ASSET=analytic -oo RANDOM_ACCESS=NO out.tif
```

7.68.1.7 See Also

- *General documentation page for PLScenes driver*
- Documentation of Planet Scenes Data API v1
- *Raster PLMosaic / Planet Mosaics API driver*

7.68.2 See Also

- Documentation of Planet Scenes Data API V1
- *Raster PLMosaic / Planet Mosaics API driver*

7.69 PMTiles

Added in version 3.8.

Driver short name

PMTiles

Driver built-in by default

This driver is built-in by default

This driver supports reading and writing PMTiles datasets containing vector tiles, encoded in the Mapbox Vector Tiles (MVT) format.

PMTiles is a single-file archive format for tiled data. A PMTiles archive can be hosted on a commodity storage platform such as S3, and enables low-cost, zero-maintenance map applications that are "serverless" - free of a custom tile backend or third party provider.

This driver is compatible with all GDAL *network-based virtual file systems*

The SRS is always the Pseudo-Mercator (a.k.a Google Mercator) projection, EPSG:3857.

The supported datasets must contain a JSON metadata document following the MBTiles specification, containing at least the `vector_layers` array.

Note that the driver will make no effort of stitching together geometries for linear or polygonal features that overlap several tiles. An application that wishes to eliminate those interrupts could potentially use the CLIP=NO open option to get larger boundaries, and use appropriate clipping graphic primitives to hide those discontinuities.

The driver also supports raster layers. See *PMTiles raster driver*

7.69.1 Driver capabilities

Supports Create()

This driver supports the *`GDALDriver::Create()`* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.69.2 Opening options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are available:

- **CLIP=[YES/NO]:** Defaults to `YES`. Whether to clip geometries of vector features to tile extent. Vector tiles are generally produced with a buffer that provides overlaps between adjacent tiles, and can be used to display them properly. When using vector tiles as a vector layer source, like in OGR vector model, this padding is undesirable, hence the default behavior of clipping.
- **ZOOM_LEVEL=<integer>:** Zoom level to use. Must be between the minimum and maximum zoom levels reported in the metadata. Defaults to the maximum zoom level available in the dataset.
- **ZOOM_LEVEL_AUTO=[YES/NO]:** Defaults to `NO`. Whether to auto-select the zoom level for vector layers according to the spatial filter extent. Only for display purpose.
- **JSON_FIELD=[YES/NO]:** Defaults to `NO`. Whether tile attributes should be serialized in a single `json` field as JSON. This may be useful if tiles may have different attribute schemas.

7.69.3 Creation issues

Tiles are generated with WebMercator (EPSG:3857) projection. Several layers can be written. It is possible to decide at which zoom level ranges a given layer is written.

Part of the conversion is multi-threaded by default, using as many threads as there are cores. The number of threads used can be controlled with the *`GDAL_NUM_THREADS`* configuration option.

The driver implements also a direct translation mode when using **`ogr2ogr`** with a MBTiles vector dataset as input and a PMTiles output dataset, without any argument: `ogr2ogr out.pmtiles in.mbtiles`. In that mode, existing MVT tiles from the MBTiles files are used as such, contrary to the general writing mode that will involve computing them by discretizing geometry coordinates.

7.69.4 Dataset creation options

Dataset creation options can be specified in command-line tools using the syntax `-dsco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following dataset creation options are supported:

- **NAME=value:** Tileset name. Defaults to the basename of the output file/directory. Used to fill metadata records.
- **DESCRIPTION=value:** A description of the tileset. Used to fill metadata records.
- **TYPE=[overlay/baselayer]:** Layer type. Used to fill metadata records.
- **MINZOOM=<integer>:** Defaults to `0`. Minimum zoom level at which tiles are generated.
- **MAXZOOM=<integer>:** Defaults to `5`. Maximum zoom level at which tiles are generated. Maximum supported value is 22.

- **CONF=[<json>/<filename>]:** Layer configuration as a JSON serialized string. Or filename containing the configuration as JSON.
- **SIMPLIFICATION=float:** Simplification factor for linear or polygonal geometries. The unit is the integer unit of tiles after quantification of geometry coordinates to tile coordinates. Applies to all zoom levels, unless *SIMPLIFICATION_MAX_ZOOM* is also defined.
- **SIMPLIFICATION_MAX_ZOOM=<float>:** Simplification factor for linear or polygonal geometries, that applies only for the maximum zoom level.
- **EXTENT=<positive integer>:** Defaults to `4096`. Number of units in a tile. The greater, the more accurate geometry coordinates (at the expense of tile byte size).
- **BUFFER=<positive integer>:** Number of units for geometry buffering. This value corresponds to a buffer around each side of a tile into which geometries are fetched and clipped. This is used for proper rendering of geometries that spread over tile boundaries by some rendering clients. Defaults to 80 if *EXTENT=4096*.
- **MAX_SIZE=<integer>:** Defaults to `500000`. Maximum size of a tile in bytes (after compression). If a tile is greater than this threshold, features will be written with reduced precision, or discarded.
- **MAX_FEATURES=<integer>:** Defaults to `200000`. Maximum number of features per tile.

7.69.5 Layer configuration

The above mentioned CONF dataset creation option can be set to a string whose value is a JSON serialized document such as the below one:

```
{
    "boundaries_lod0": {
        "target_name": "boundaries",
        "description": "Country boundaries",
        "minzoom": 0,
        "maxzoom": 2
    },
    "boundaries_lod1": {
        "target_name": "boundaries",
        "minzoom": 3,
        "maxzoom": 5
    }
}
```

boundaries_lod0 and *boundaries_lod1* are the name of the OGR layers that are created into the target MVT dataset. They are mapped to the MVT target layer *boundaries*.

It is also possible to get the same behavior with the below layer creation options, although that is not convenient in the ogr2ogr use case.

7.69.6 Layer creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALDatasetCreateLayer()* (C) or *Dataset.CreateLayer* (Python). The following layer creation options are supported:

- **MINZOOM=<integer>:** Minimum zoom level at which tiles are generated. Defaults to the dataset creation option *MINZOOM* value.
- **MAXZOOM=<integer>:** Maximum zoom level at which tiles are generated. Defaults to the dataset creation option *MAXZOOM* value. Maximum supported value is 22.

- **NAME=value:** Target layer name. Defaults to the layer name, but can be overridden so that several OGR layers map to a single target layer. The typical use case is to have different OGR layers for mutually exclusive zoom level ranges.
- **DESCRIPTION=value:** A description of the layer.

7.69.7 /vsipmtiles/ virtual file system

The /vsipmtiles/ virtual file system offers a view of the content of a PMTiles dataset has a file hierarchy, with the following structure:

```
/pmtiles_header.json: JSON view of the PMTiles header
/metadata.json: JSON metadata document stored in the dataset
/{z}/: Directory with tiles for zoom level z
/{z}/{x}/: Directory with tiles for zoom level z and x
/{z}/{x}/{y}.{ext}: Tile data
```

gdal vsi list and *gdal vsi copy* utilities can be used to explore and extract data from a PMTiles dataset

Listing the content of a dataset:

```
gdal vsi list -lR "/vsipmtiles//vsicurl/https://protomaps.github.io/PMTiles/
↪protomaps(vector)ODbL_firenze.pmtiles"
```

outputs:

```
-r--r--r--  1 unknown unknown          809 2023-05-29 09:06 /vsipmtiles//vsicurl/https://
↪protomaps.github.io/PMTiles/protomaps(vector)ODbL_firenze.pmtiles/pmtiles_header.json
-r--r--r--  1 unknown unknown         1872 2023-05-29 09:06 /vsipmtiles//vsicurl/https://
↪protomaps.github.io/PMTiles/protomaps(vector)ODbL_firenze.pmtiles/metadata.json
dr-xr-xr-x  1 unknown unknown            0 2023-05-29 09:06 /vsipmtiles//vsicurl/https://
↪protomaps.github.io/PMTiles/protomaps(vector)ODbL_firenze.pmtiles/0/
dr-xr-xr-x  1 unknown unknown            0 2023-05-29 09:06 /vsipmtiles//vsicurl/https://
↪protomaps.github.io/PMTiles/protomaps(vector)ODbL_firenze.pmtiles/0/0/
-r--r--r--  1 unknown unknown          588 2023-05-29 09:06 /vsipmtiles//vsicurl/https://
↪protomaps.github.io/PMTiles/protomaps(vector)ODbL_firenze.pmtiles/0/0/0.mvt
dr-xr-xr-x  1 unknown unknown            0 2023-05-29 09:06 /vsipmtiles//vsicurl/https://
↪protomaps.github.io/PMTiles/protomaps(vector)ODbL_firenze.pmtiles/1/
dr-xr-xr-x  1 unknown unknown            0 2023-05-29 09:06 /vsipmtiles//vsicurl/https://
↪protomaps.github.io/PMTiles/protomaps(vector)ODbL_firenze.pmtiles/1/1/
-r--r--r--  1 unknown unknown          590 2023-05-29 09:06 /vsipmtiles//vsicurl/https://
↪protomaps.github.io/PMTiles/protomaps(vector)ODbL_firenze.pmtiles/1/1/0.mvt
[ ... snip ... ]
-r--r--r--  1 unknown unknown          771 2023-05-29 09:06 /vsipmtiles//vsicurl/https://
↪protomaps.github.io/PMTiles/protomaps(vector)ODbL_firenze.pmtiles/14/8707/5974.mvt
```

Displaying the metadata JSON file:

```
gdal vsi copy "/vsipmtiles//vsicurl/https://protomaps.github.io/PMTiles/
↪protomaps(vector)ODbL_firenze.pmtiles/metadata.json" /vsistdout/ | jq .
```

outputs:

```
{
  "attribution": "<a href=\"https://protomaps.com\" target=\"_blank\">Protomaps</a> © <a␣
```

(continues on next page)

(continued from previous page)

```
↪href=\"https://www.openstreetmap.org\" target=\"_blank\"> OpenStreetMap</a>",
  "name": "protomaps 2023-01-18T07:49:39Z",
  "type": "baselayer",
  "vector_layers": [
    {
      "fields": {},
      "id": "earth"
    },
    {
      "fields": {
        "boundary": "string",
        "landuse": "string",
        "leisure": "string",
        "name": "string",
        "natural": "string"
      },
      "id": "natural"
    },
    { "... snip ...": {} },
    {
      "fields": {
        "pmap:min_admin_level": "number"
      },
      "id": "boundaries"
    },
    {
      "fields": {},
      "id": "mask"
    }
  ]
}
```

Extracting all content in a local directory:

```
gdal vsi copy -r "/vsipmtiles//vsicurl/https://protomaps.github.io/PMTiles/
↪protomaps(vector)ODbL_firenze.pmtiles" out_pmtiles
```

7.69.8 Examples

7.69.8.1 Example 1: Simple translation of a single shapefile into PMTiles

Dataset creation options (dsco) MINZOOM and MAXZOOM specifies tile zoom levels.

```
ogr2ogr -dsco MINZOOM=10 -dsco MAXZOOM=20 -f "PMTiles" filename.pmtiles my_shapes.shp
```

7.69.8.2 Example 2: Merge all PostgreSQL/PostGIS tables in a schema into a single PMTiles file

PostgreSQL table names are used as layer names. Dataset creation options (dsco) MINZOOM and MAXZOOM specifies tile zoom levels.

```
ogr2ogr -dsco MINZOOM=0 -dsco MAXZOOM=22 -f "PMTiles" filename.pmtiles "PG:host=my_host␣
↪port=my_port dbname=my_database user=my_user password=my_password schemas=my_schema"
```

7.69.9 See Also

- *PMTiles raster driver*
- PMTiles specification
- *Mapbox Vector tiles driver*
- *MBTiles driver*

7.70 IHO S-57 (ENC)

Driver short name

S57

Driver built-in by default

This driver is built-in by default

International Hydrographic Organisation (IHO) S-57 Electronic Navigation Charts (ENC) datasets are supported for read access.

The S-57 driver module produces features for all S-57 features in the S-57 file, and associated updates. S-57 (ENC) files normally have the extension ".000".

S-57 feature objects are translated into features. S-57 geometry objects are automatically collected and formed into geometries on the features.

The S-57 reader depends on having two supporting files, s57objectclasses.csv, and s57attributes.csv available at runtime in order to translate features in an object class specific manner. These should be in the directory pointed to by the environment variable S57_CSV, or in the current working directory.

S-57 update files contain information on how to update a distributed S-57 base data file. The base files normally have the extension .000 while the update files have extensions like .001, .002 and so on. The S-57 reader will normally read and apply all updates files to the in memory version of the base file on the fly. The feature data provided to the application therefore includes all the updates.

7.70.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.70.2 Feature Translation

Normally all features read from S-57 are assigned to a layer based on the name of the object class (OBJL) to which they belong. For instance, with an OBJL value of 2, the feature is an "Airport / airfield" and has a short name of "AIRARE" which is used as the layer name. A typical S-57 transfer will have in excess of 100 layers.

Each feature type has a predefined set of attributes as defined by the S-57 standard. For instance, the airport (AIRARE) object class can have the AIRARE, CATAIR, CONDTN, CONVIS, NOBJNM, OBJNAM, STATUS, INFORM, NINFOM, NTXTDS, PICREP, SCAMAX, SCAMIN, TXTDSC, RECDAT, RECIND, SORDAT, and SORIND attributes. These short names can be related to longer, more meaningful names using an S-57 object/attribute catalog such as the S-57 standard document itself, or the catalog files (s57attributes.csv, and s57objectclasses.csv). Such a catalog can also be used to establish all the available object classes, and their attributes.

The following are some common attributes, including generic attributes which appear on all feature, regardless of object class. is turned on.

```
Attribute Name  Description                          Defined On
--------------  -----------                          ----------

GRUP            Group number.                        All features

OBJL            Object label code.  This number      All features
                indicates the object class of the
                feature.

RVER            Record version.

AGEN            Numeric agency code, such as 50 for  All features
                the Canadian Hydrographic Service.

FIDN            Feature identification number.       All features

FIDS            Feature identification subdivision.  All features

INFORM          Informational text.                  Some features

NINFOM          Informational text in national       Some features
                language.

OBJNAM          Object name                          Some features

NOBJNM          Object name in national              Some features
                language.

SCAMAX          Maximum scale for display            Some features

SCAMIN          Minimum scale for display            Some features

SORDAT          Source date                          Some features
```

The following are present if LNAM_REFS is enabled:

```
LNAM            Long name.  An encoding of AGEN,       All features
                FIDN and FIDS used to uniquely
                identify this features within an
                S-57 file.

LNAM_REFS       List of long names of related features All Features

FFPT_RIND       Relationship indicators for each of    All Features
                the LNAM_REFS relationships.
```

7.70.3 DSID layer

Dataset wide fields, such as DSID (Data Set Identification), DSSI (Data Set Structure Information) and DSPM (Data Set Parameter) are exposed in a layer `DSID` which has a single feature. See paragraph 7.3.1 "Data set general information record structure" of IHO S-57 Edition 3.1 standard (main)

7.70.4 Soundings

Depth soundings are handled somewhat specially in S-57 format, in order to efficiently represent the many available data points. In S-57 one sounding feature can have many sounding points. The S-57 reader splits each of these out into its own feature type `SOUNDG' feature with an s57_type of `s57_point3d'. All the soundings from a single feature record will have the same AGEN, FIDN, FIDS and LNAM value.

7.70.5 S57 Control Options

There are several control options which can be used to alter the behavior of the S-57 reader. Users can set these by appending them in the OGR_S57_OPTIONS environment variable.

They can also be specified independently as open options to the driver. Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are supported:

- **UPDATES=[APPLY/IGNORE]:** Defaults to `APPLY`. Should update files be incorporated into the base data on the fly.
- **SPLIT_MULTIPOINT=[ON/OFF]:** Defaults to `OFF`. Should multipoint soundings be split into many single point sounding features. Multipoint geometries are not well handle by many formats, so it can be convenient to split single sounding features with many points into many single point features.
- **ADD_SOUNDG_DEPTH=[ON/OFF]:** Defaults to `OFF`. Should a DEPTH attribute be added on SOUNDG features and assign the depth of the sounding. This should only be enabled with SPLIT_MULTIPOINT is also enabled.
- **RETURN_PRIMITIVES=[ON/OFF]:** Defaults to `OFF`. Should all the low level geometry primitives be returned as special IsolatedNode, ConnectedNode, Edge and Face layers. Note that for features of the Edge layer, the returned OGR LineString geometry does not include the start and end nodes. Their coordinates can be retrieved by joining with the NAME_RCID_0 (identifier of the start node) and NAME_RCID_1 (identifier of the end node) with the RCID field of the ConnectedNode layer.
- **PRESERVE_EMPTY_NUMBERS=[ON/OFF]:** Defaults to `OFF`. If enabled, numeric attributes assigned an empty string as a value will be preserved as a special numeric value. This option should not generally be needed, but may be useful when translated S-57 to S-57 losslessly.
- **LNAM_REFS=[ON/OFF]:** Defaults to `OFF`. Should LNAM and LNAM_REFS fields be attached to features capturing the feature to feature relationships in the FFPT group of the S-57 file.

- **RETURN_LINKAGES=[ON/OFF]:** Defaults to `OFF`. Should additional attributes relating features to their underlying geometric primitives be attached. These are the values of the FSPT group, and are primarily needed when doing S-57 to S-57 translations.
- **RECODE_BY_DSSI=[ON/OFF]:** Defaults to `ON`. Should attribute values be recoded to UTF-8 from the character encoding specified in the S57 DSSI record. Default is ON starting with GDAL 3.4.1.
- **LIST_AS_STRING=[ON/OFF]:** (GDAL >= 3.2) Defaults to `OFF`. Whether attributes tagged as list in S57 dictionaries should be reported as a String field, instead of a StringList. Before GDAL 3.2, the behavior was equivalent to setting this option to ON.

Example:

```
set OGR_S57_OPTIONS = "RETURN_PRIMITIVES=ON,RETURN_LINKAGES=ON,LNAM_REFS=ON"
```

7.70.6 S-57 Export

Preliminary S-57 export capability is intended only for specialized use, and is not properly documented at this time. Setting the following options is a minimum required to support S-57 to S-57 conversion via OGR.

```
set OGR_S57_OPTIONS = "RETURN_PRIMITIVES=ON,RETURN_LINKAGES=ON,LNAM_REFS=ON"
```

Dataset creation options can be specified in command-line tools using the syntax `-dsco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following dataset creation options are supported to supply basic information for the S-57 data set descriptive records (DSID and DSPM, see the S-57 standard for a more detailed description):

- **S57_EXPP=value:** Defaults to `1`. Exchange purpose.
- **S57_INTU=value:** Defaults to `4`. Intended usage.
- **S57_EDTN=value:** Defaults to `2`. Edition number.
- **S57_UPDN=value:** Defaults to `0`. Update number.
- **S57_UADT=value:** Defaults to `200308081`. Update application date.
- **S57_ISDT=value:** Defaults to `200308081`. Issue date.
- **S57_STED=value:** Defaults to `03.1`. Edition number of S-57.
- **S57_AGEN=value:** Defaults to `540`. Producing agency.
- **S57_COMT:=value:** Comment.
- **S57_AALL=value:** Defaults to `0`. Lexical level used for the ATTF fields.
- **S57_NALL=value:** Defaults to `0`. Lexical level used for the NATF fields.
- **S57_NOMR=value:** Defaults to `0`. Number of meta records (objects with acronym starting with "M_").
- **S57_NOGR=value:** Defaults to `0`. Number of geo records.
- **S57_NOLR=value:** Defaults to `0`. Number of collection records.
- **S57_NOIN=value:** Defaults to `0`. Number of isolated node records.
- **S57_NOCN=value:** Defaults to `0`. Number of connected node records.
- **S57_NOED=value:** Defaults to `0`. Number of edge records.
- **S57_HDAT=value:** Defaults to `2`. Horizontal geodetic datum.
- **S57_VDAT=value:** Defaults to `17`. Vertical datum.

- **S57_SDAT=value:** Defaults to `23`. Sounding datum.
- **S57_CSCL=value:** Defaults to `52000`. Compilation scale of data (1:X).
- **S57_COMF=value:** Defaults to `10000000`. Floating-point to integer multiplication factor for coordinate values.
- **S57_SOMF=value:** Defaults to `10`. Floating point to integer multiplication factor for 3-D (sounding) values.

7.70.7 See Also

- S-57 Online Object/Attribute Catalog
- Frank's S-57 Page (at archive.org): Links to other resources, and sample datasets.
- IHO S-57 Edition 3.1 standard (main)
- S-57 Appendix B
- *IHO S-101 (ENC)*

7.71 IHO S-101 (ENC)

Added in version 3.14.

Driver short name

S101

Driver built-in by default

This driver is built-in by default

International Hydrographic Organization (IHO) S-101 Electronic Navigation Charts (ENC) datasets are supported for read access.

S-101 (ENC) files normally start with "101" and have the extension ".000".

The S-101 driver produces OGR features for all S-101 information (in a broad sense, this includes geometry primitives, Information Type, and Feature Type records) in the S-101 file.

Georeferenced features are referenced to WGS 84 (EPSG:4326). Point and multipoint geometries may be referenced to a vertical datum (depth, in meters).

S-101 update files contain information on how to update a distributed S-101 base data file. The base files normally have the extension .000 while the update files have extensions like .001, .002 and so on. The S-101 reader will normally read and apply all update files to the in memory version of the base file on the fly. The feature data provided to the application therefore includes all the updates.

7.71.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.71.2 Open Options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are supported:

- **STRICT=[YES/NO]:** Defaults to `YES`. Whether strict parsing mode should be enabled. Strict mode validates the product against the specification and errors out as soon as an error is encountered. When setting this option to NO, warnings will be emitted instead and the parsing of the product will continue as much as possible.
- **UPDATES=[APPLY/IGNORE]:** Defaults to `APPLY`. Should update files be incorporated into the base data on the fly.

7.71.3 Layers

The following layers are exposed (when there are corresponding records in the dataset):

7.71.3.1 `informationType` layer

No geometry field.

One OGR feature per Information Type record.

The following OGR fields are available:

- `recordId`, Integer: content of the `RCID` subfield of the `IRID` field.
- `recordVersion`, Integer: content of the `RVER` subfield of the `IRID` field.
- `informationType`, String: type of the record (e.g., `ContactDetails`, `SpatialQuality`, etc.).
- Attributes of the `ATTR` field, exposed in a flattened way (e.g., `spatialAccuracy.horizontalPositionUncertainty.uncertaintyFixed`).

When one information association is present, the following optional fields are also available:

- `infoAssociationRecordId`, Integer: foreign key to the `recordId` value of another informationType feature.
- `infoAssociationCode`, String: type of information association (e.g., `AdditionalInformation`).
- `infoAssociationRoleCode`, String: role of the association (e.g., `providesInformation`, `supports`, etc.).
- Attributes of the association, exposed in a flattened way and prefixed with `association_`.

When several information associations are present, the following optional fields are also available:

- `infoAssociationRecordId[{idx}]`, Integer: foreign key to the `recordId` value of another informationType feature.
- `infoAssociationCode[{idx}]`, String: type of information association (e.g., `AdditionalInformation`).
- `infoAssociationRoleCode[{idx}]`, String: role of the association (e.g., `providesInformation`, `supports`, etc.).
- Attributes of the association, exposed in a flattened way and prefixed with `association[{idx}]_`.

7.71.3.2 `Point2D` layer

Geometry field: 2D point in WGS 84.

The following OGR fields are available:

- `recordId`, Integer: content of the `RCID` subfield of the `PRID` field.
- `recordVersion`, Integer: content of the `RVER` subfield of the `PRID` field.

Optional information association(s) are exposed in the same way as for the `informationType` layer.

7.71.3.3 `Point3D_{vertical_crs_name}` layers

Geometry field: 3D point in a compound CRS, whose horizontal part is WGS 84 and vertical part is a depth.

Same fields as the `Point2D` layer.

7.71.3.4 `MultiPoint2D` layer

Geometry field: 2D multipoint in WGS 84.

The following OGR fields are available:

- `recordId`, Integer: content of the `RCID` subfield of the `MRID` field.
- `recordVersion`, Integer: content of the `RVER` subfield of the `MRID` field.

Optional information association(s) are exposed in the same way as for the `informationType` layer.

7.71.3.5 `MultiPoint3D_{vertical_crs_name}` layers

Geometry field: 3D point in a compound CRS, whose horizontal part is WGS 84 and vertical part is a depth.

Same fields as the `MultiPoint2D` layer.

7.71.3.6 `Curve` layer

Geometry field: 2D linestring in WGS 84.

The following OGR fields are available:

- `recordId`, Integer: content of the `RCID` subfield of the `CRID` field.
- `recordVersion`, Integer: content of the `RVER` subfield of the `CRID` field.

Optional information association(s) are exposed in the same way as for the `informationType` layer.

7.71.3.7 `CompositeCurve` layer

Geometry field: 2D linestring in WGS 84.

The following OGR fields are available:

- `recordId`, Integer: content of the `RCID` subfield of the `CCID` field.
- `recordVersion`, Integer: content of the `RVER` subfield of the `CCID` field.

Optional information association(s) are exposed in the same way as for the `informationType` layer.

7.71.3.8 `Surface` layer

Geometry field: 2D polygon in WGS 84.

The following OGR fields are available:

- `recordId`, Integer: content of the `RCID` subfield of the `SRID` field.
- `recordVersion`, Integer: content of the `RVER` subfield of the `SRID` field.

Optional information association(s) are exposed in the same way as for the `informationType` layer.

7.71.3.9 Feature type layers

One layer per feature type (as determined by the `NTFC` subfield of the `FRID` field) and per geometry type.

The layer name is `{featureTypeShortName}_{geometryType}`, e.g., `AnchorageArea_Point2D` and `AnchorageArea_Polygon`, `RangeSystem_NoGeom`.

If several geometries of the same nature are associated with at least one feature, multi-geometry types are used (MultiPoint, MultiLineString, MultiPolygon, or GeometryCollection when the individual geometry is a MultiPoint).

The following OGR fields are available:

- `recordId`, Integer: content of the `RCID` subfield of the `FRID` field.
- `recordVersion`, Integer: content of the `RVER` subfield of the `FRID` field.
- `featureIdentificationNumber`, Integer: content of the `FIDN` subfield of the `FOID` field.
- `featureIdentificationSubdivision`, Integer: content of the `FIDS` subfield of the `FOID` field.
- `producingAgency`, Integer: content of the `AGEN` subfield of the `FOID` field. The corresponding text name is in the IHO S-62 Producer Code Register.

For spatial layers, the following additional fields are available:

- `geometryLayerName`, String (or StringList for multi-geometries): name of the layer of the associated geometry.
- `geometryRecordId`, Integer (or IntegerList for multi-geometries): foreign key to the `recordId` value of the associated geometry in layer `geometryLayerName`.
- `geometryOrientation`, String (or StringList for multi-geometries): `forward` if the associated curve must be followed in its nominal orientation, or `reverse` if it must be followed in the reverse direction of its definition. Only present when the geometry is a `Curve` or `CompositeCurve`.
- `scaleMinimum`, Integer (or IntegerList for multi-geometries): the smallest (reciprocal) scale at which a feature is intended to be displayed.
- `scaleMaximum`, Integer (or IntegerList for multi-geometries): the largest (reciprocal) scale at which a feature is intended to be displayed.

Optional information association(s) are exposed in the same way as for the `informationType` layer.

Optional feature association(s) are exposed in a similar way as information association(s), but using a `featureAssociation` field name prefix instead of `informationType`.

In certain circumstances, the symbolization of a curve may need to be suppressed in surface features. When masking is defined, the following additional fields are available:

- `maskLayerName`, String (or StringList for multi-mask geometries): name of the layer of the associated geometry (`Curve` or `CompositeCurve`).
- `maskRecordId`, Integer (or IntegerList for multi-mask geometries): foreign key to the `recordId` value of the associated geometry in layer `maskLayerName`.

- `maskIndicator`, String (or StringList for multi-mask geometries): how the mask must be applied. The two potential values are `truncatedByDataCoverageLimit` or `suppressPortrayal`.

The mask geometry is also attached as an OGR geometry in an additional `maskGeometry` OGR geometry field. Note that upon file format conversion, this additional geometry field may be lost due to file format limitations. The `maskLayerName` and `maskRecordId` attributes may then be used to recover the mask geometry.

7.71.4 Dataset metadata

The following metadata items are reported at the dataset level:

- `APPLICATION_PROFILE`. 1 if only initial file .000 processed, 2 otherwise.
- `DATASET_EDITION`. Dataset version number. Set to 0 if dataset is cancelled
- `DATASET_IDENTIFIER`. The filename, including extension but excluding any path information.
- `DATASET_LANGUAGE`. Primary language used in the dataset. Must be `EN` (English).
- `DATASET_REFERENCE_DATE`. Reference date of the dataset. Format: YYYYMMDD according to ISO 8601.
- `DATASET_TITLE`. Title of the dataset.
- `ENCODING_SPECIFICATION`. Must be `S-100 Part 10a`.
- `ENCODING_SPECIFICATION_EDITION`. Must be `5.2`.
- `PRODUCT_EDITION`. Must be `2.0`.
- `PRODUCT_IDENTIFIER`. Must be `INT.IHO.S-101.2.0`.
- `STATUS`. `VALID` or `CANCELLED`.

7.71.5 Feature Catalog

The driver can operate without the XML Feature Catalog file being present on the system, but access to that file provides:

- better typing of attributes, in particular boolean fields.
- extra consistency checks on the validity of attributes and feature types.
- exposing enumerated attributes as OGR field domains.

The feature catalog is looked for in the following locations, by decreasing priority:

- Value defined by the `GDAL_S101_FEATURE_CATALOG` configuration option.
- In `$INSTALL_PREFIX/share/gdal/101_Feature_Catalogue_2.0.0.xml`.
- In `$HOME/.gdal/101_Feature_Catalogue_2.0.0.xml`.

If not found, an attempt is made to download it from the IHO OHI GitHub repository and cache it in `$HOME/.gdal/101_Feature_Catalogue_2.0.0.xml`.

7.71.6 See Also

- IHO S-101 Electronic Navigational Chart (ENC) Product Specification - Edition 2.0.0
- *IHO S-57 (ENC)*

7.71.7 Credits

Initial development of this driver was funded by Rijkswaterstaat. Support for reading update files was funded by Zydromarine.

7.72 Selafin files

Driver short name

Selafin

Driver built-in by default

This driver is built-in by default

OGR supports reading from 2D Selafin/Seraphin files. Selafin/Seraphin is the generic output and input format of geographical files in the open-source Telemac hydraulic model. The file format is suited to the description of numerical attributes for a set of point features at different time steps. Those features usually correspond to the nodes in a finite-element model. The file also holds a connectivity table which describes the elements formed by those nodes and which can also be read by the driver.

The driver supports the use of VSI virtual files as Selafin datasources.

The driver offers full read-write support on Selafin files. However, due to the particular nature of Selafin files where element (polygon) features and node (point) features are closely related, writing on Selafin layers can lead to counter-intuitive results. In a general way, writing on any layer of a Selafin data-source will cause side effects on all the other layers. Also, it is very important **not to open the same datasource more than once in update mode**. Having two processes write at the same time on a single datasource can lead to irreversible data corruption. The driver issues a warning each time a datasource is opened in update mode.

7.72.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.72.2 Magic bytes

There is no generic extension to Selafin files. The adequate format is tested by looking at a dozen of magic bytes at the beginning of the file:

- The first four bytes of the file should contain the values (in hexadecimal): 00 00 00 50. This actually indicates the start of a string of length 80 in the file.
- At position 84 in the file, the eight next bytes should read (in hexadecimal): 00 00 00 50 00 00 00 04.

Files which match those two criteria are considered to be Selafin files and the driver will report it has opened them successfully.

7.72.3 Format

Selafin format is designed to hold data structures in a portable and compact way, and to allow efficient random access to the data. To this purpose, Selafin files are binary files with a generic structure.

7.72.3.1 Elements

Selafin files are made of the juxtaposition of elements. Elements have one of the following types:

- integer,
- string,
- floating point values,
- arrays of integers,
- arrays of floating point values.

Element	Internal representation	Comments
Integer	a b c d	Integers are stored on 4 bytes in big-endian format (most significant byte first). The value of the integer is 2^{24}.a+2:su p:*16*.b+2^{8}. c+d.
Floating point	a b c d	Floating point values are stored on 4 bytes in IEEE 754 format and under big-endian convention (most significant byte first). Endianness is detected at run time only once when the first floating point value is read.
String	Length 1 2 ... Length	Strings are stored in three parts: • an integer holding the length (in characters) of the string, over 4 bytes; • the sequence of characters of the string, each character on one byte; • the same integer with the length of the string repeated
Array of integers	Length 1 2 ... Length	Arrays of integers are stored in three parts: • an integer holding the length (in bytes, thus 4 times the number of elements) of the array, over 4 bytes; • the sequence of integers in the array, each integer on 4 bytes as described earlier; • the same integer with the length of the array repeated
Array of floating point values	Length 1 2 ... Length	Arrays of floating point values are stored in three parts: • an integer holding the length (in bytes, thus 4 times the number of elements) of the array, over 4 bytes; • the sequence of floating point values in the array, each one on 4 bytes as described earlier; • the same integer with the length of the array repeated

7.72.3.2 Full structure

The header of a Selafin file holds the following elements in that exact order:

- a *string* of 80 characters with the title of the study; the last 8 characters shall be "SERAPHIN" or "SERAFIN" or "SERAFIND";

- an *array of integers* of exactly 2 elements, the first one being the number of variables (attributes) *nVar*, and the second is ignored;
- *nVar strings* with the names of the variables, each one with length 32;
- an *array of integers* of exactly 10 elements:
  - the third element is the x-coordinate of the origin of the model;
  - the fourth element is the y-coordinate of the origin of the model;
  - the tenth element *isDate* indicates if the date of the model has to be read (see later);
  - in addition, the second element being unused by hydraulic software at the moment, it is used by the driver to store the spatial reference system of the datasource, in the form of a single integer with the EPSG number of the projection;
- if *isDate*=1, an *array of integers* of exactly 6 elements, with the starting date of the model (year, month, day, hour, minute, second);
- an *array of integers* of exactly 4 elements:
  - the first element is the number of elements *nElements*,
  - the second element is the number of points *nPoints*,
  - the third element is the number of points per element*nPointsPerElement*,
  - the fourth element must be 1;
- an *array of integers* of exactly *nElements*nPointsPerElement* elements, with each successive set of *nPointsPerElement* being the list of the number of the points (number starting with 1) constituting an element;
- an *array of integers* of exactly *nPoints* elements ignored by the driver (the elements shall be 0 for inner points and another value for the border points where a limit condition is applied);
- an *array of floating point values* of exactly *nPoints* elements with the x-coordinates of the points;
- an *array of floating point values* of exactly *nPoints* elements with the y-coordinates of the points;

The rest of the file actually holds the data for each successive time step. A time step contains the following elements:

- a *array of floating point values* of exactly 1 element, being the date of the time step relative to the starting date of the simulation (usually in seconds);
- *nVar array of floating point values*, each with exactly *nPoints* elements, with the values of each attribute for each point at the current time step.

7.72.4 Mapping between file and layers

7.72.4.1 Layers in a Selafin datasource

The Selafin driver accepts only Selafin files as data sources.

Each Selafin file can hold one or several time steps. All the time steps are read by the driver and two layers are generated for each time step:

- one layer with the nodes (points) and their attributes: its name is the base name of the data source, followed by "_p" (for Points);
- one layer with the elements (polygons) and their attributes calculated as the averages of the values of the attributes of their vertices: its name is the base name of the data source, followed by "_e" (for Elements).

Finally, either the number of the time step, or the calculated date of the time step (based on the starting date and the number of seconds elapsed), is added to the name. A data source in a file called Results may therefore be read as several layers:

- `Results_p2014_05_01_20_00_00`, meaning that the layers holds the attributes for the nodes and that the results hold for the time step at 8:00 PM, on May 1st, 2014;
- `Results_e2014_05_01_20_00_00`, meaning that the layers holds the attributes for the elements and that the results hold for the time step at 8:00 PM, on May 1st, 2014;
- `Results_p2014_05_01_20_15_00`, meaning that the layers holds the attributes for the elements and that the results hold for the time step at 8:15 PM, on May 1st, 2014;
- ...

7.72.4.2 Constraints on layers

Because of the *format of the Selafin file*, the layers in a single Selafin datasource are not independent from each other. Changing one layer will most certainly have side effects on all other layers. The driver updates all the layers to match the constraints:

- All the point layers have the same number of features. When a feature is added or removed in one layer, it is also added or removed in all other layers.
- Features in different point layers share the same geometry. When the position of one point is changed, it is also changed in all other layers.
- All the element layers have the same number of features. When a feature is added or removed in one layer, it is also added or removed in all other layers.
- All the polygons in element layers have the same number of vertices. The number of vertices is fixed when the first feature is added to an element layer, and can not be changed afterwards without recreating the datasource from scratch.
- Features in different element layers share the same geometry. When an element is added or removed in one layer, it is also added or removed in all other layers.
- Every vertex of every feature in an element layer has a corresponding point feature in the point layers. When an element feature is added, if its vertices do not exist yet, they are created in the point layers.
- Points and elements layers only support attributes of type "REAL". The format of real numbers (width and precision) can not be changed.

7.72.5 Layer filtering specification

As a single Selafin files may hold millions of layers, and the user is generally interested in only a few of them, the driver supports syntactic sugar to filter the layers before they are read.

When the datasource is specified, it may be followed immediately by a *layer filtering specification.*, as in `Results[0:10]`. The effects of the layer filtering specification is to indicate which time steps shall be loaded from all Selafin files.

The layer filtering specification is a comma-separated sequence of range specifications, delimited by square brackets and maybe preceded by the character 'e' or 'p'. The effect of characters 'e' and 'p' is to select respectively either only elements or only nodes. If no character is added, both nodes and elements are selected. Each range specification is:

- either a single number, representing one single time step (whose numbers start with 0),
- or a set of two numbers separated by a colon: in that case, all the time steps between and including those two numbers are selected; if the first number is missing, the range starts from the beginning of the file (first time step); if the second number is missing, the range goes to the end of the file (last time step);

Numbers can also be negative. In this case, they are counted from the end of the file, -1 being the last time step.

Some examples of layer filtering specifications:

[0]	First time step only, but return both points and elements
[e:9]	First 10 time steps only, and only layers with elements
[p-4:]	Last 4 time steps only, and only layers with nodes
[3,10,-2:-1]	4th, 11th, and last two time steps, for both nodes and elements

7.72.6 Datasource creation options

Dataset creation options can be specified in command-line tools using the syntax `-dsco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following dataset creation options are supported:

- **TITLE=value:** Title of the datasource, stored in the Selafin file. The title must not hold more than 72 characters. If it is longer, it will be truncated to fit in the file.
- **DATE=value:** Starting date of the simulation. Each layer in a Selafin file is characterized by a date, counted in seconds since a reference date. This option allows providing the reference date. The format of this field must be YYYY-MM-DD_hh:mm:ss. The format does not mention the time zone.

An example of datasource creation option is: `-dsco TITLE="My simulation" -dsco DATE=2014-05-01_10:00:00`.

7.72.7 Layer creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALDatasetCreateLayer()`* (C) or *`Dataset.CreateLayer`* (Python). The following layer creation options are supported:

- **DATE=value:** Date of the time step relative to the starting date of the simulation (see *Datasource creation options*). This is a single floating-point value giving the number of seconds since the starting date.

An example of a layer creation option is: `-lco DATE=24000`.

7.72.8 Notes about the creation and the update of a Selafin datasource

The driver supports creating and writing to Selafin datasources, but there are some caveats when doing so.

When a new datasource is created, it does not contain any layer, feature or attribute.

When a new layer is created, it automatically inherits the same number of features and attributes as the other layers of the same type (points or elements) already in the datasource. The features inherit the same geometry as their corresponding ones in other layers. The attributes are set to 0. If there was no layer in the datasource yet, the new layer is created with no feature and attribute.In any case, when a new layer is created, two layers are actually added: one for points and one for elements.

New features and attributes can be added to the layers or removed. The behavior depends on the type of layer (points or elements). The following table explains the behavior of the driver in the different cases.

Operation	Points layers	Element layers
Change the geometry of a feature	The coordinates of the point are changed in the current layer and all other layers in the datasource.	The coordinates of all the vertices of the element are changed in the current layer and all other layers in the datasource. It is not possible to change the number of vertices. The order of the vertices matters.
Change the attribute of a feature	The attributes of the point are changed in the current layer only.	No effect.
Add a new feature	A new point is added at the end of the list of features, for this layer and all other layers. Its attributes are set to the values of the new feature.	The operation is only allowed if the new feature has the same number of vertices as the other features in the layer. The vertices are checked to see if they currently exist in the set of points. A vertex is considered equal to a point if its distance is less than some maximum distance, approximately equal to 1/1000th of the average distance between two points in the points layers. When a corresponding point is found, it is used as a vertex for the element. If no point is found, a new is created in all associated layers.
Delete a feature	The point is removed from the current layer and all point layers in the datasource. All elements using this point as a vertex are also removed from all element layers in the datasource.	The element is removed from the current layer and all element layers in the datasource.

Typically, this implementation of operations is exactly what you'll expect. For example, ogr2ogr can be used to reproject the file without changing the inner link between points and elements.

It should be noted that update operations on Selafin datasources are very slow. This is because the format does no allow for quick insertions or deletion of features and the file must be recreated for each operation.

7.72.9 VSI Virtual File System API support

The driver supports reading and writing to files managed by VSI Virtual File System API, which include "regular" files, as well as files in the /vsizip/ (read-write) , /vsigzip/ (read-only) , /vsicurl/ (read-only) domains.

7.72.10 Other notes

There is no SRS specification in the Selafin standard. The implementation of SRS is an addition of the driver and stores the SRS in an unused data field in the file. Future software using the Selafin standard may use this field and break the SRS specification. In this case, Selafin files will still be readable by the driver, but their writing will overwrite a value which may have another purpose.

7.73 ESRI Shapefile / DBF

Driver short name

ESRI Shapefile

Driver built-in by default

This driver is built-in by default

All varieties of ESRI Shapefiles should be available for reading, creation and editing. The driver can also handle standalone DBF files without associated .shp files.

Normally the OGR Shapefile driver treats a whole directory of shapefiles as a dataset, and a single shapefile within that directory as a layer. In this case the directory name should be used as the dataset name. However, it is also possible to use one of the files (.shp, .shx or .dbf) in a shapefile set as the dataset name, and then it will be treated as a dataset with one layer.

Note that when reading a Shapefile of type SHPT_ARC, the corresponding layer will be reported as of type wkbLineString, but depending on the number of parts of each geometry, the actual type of the geometry for each feature can be either OGRLineString or OGRMultiLineString. The same applies for SHPT_POLYGON shapefiles, reported as layers of type wkbPolygon, but depending on the number of parts of each geometry, the actual type can be either OGRPolygon or OGRMultiPolygon.

Measures (M coordinate) are supported. A Shapefile with measures is created if the specified geometry type is measured or an appropriate layer creation option is used. When a shapefile which may have measured geometries is opened, the first shape is examined and if it uses measures, the geometry type of the layer is set accordingly. This behavior can be changed with the ADJUST_GEOM_TYPE open option.

MultiPatch files are read and each patch geometry is turned into a TIN or a GEOMETRYCOLLECTION of TIN representation for fans and meshes.

If a .prj files in old Arc/Info style or new ESRI OGC WKT style is present, it will be read and used to associate a projection with features. A match will be attempted with the EPSG databases to identify the SRS of the .prj with an entry in the catalog.

The read driver assumes that multipart polygons follow the specification, that is to say the vertices of outer rings should be oriented clockwise on the X/Y plane, and those of inner rings counterclockwise. If a Shapefile is broken w.r.t. that rule, it is possible to define the configuration option *OGR_ORGANIZE_POLYGONS* to DEFAULT to proceed to a full analysis based on topological relationships of the parts of the polygons so that the resulting polygons are correctly defined in the OGC Simple Feature convention.

Starting with GDAL 3.13, `.shp.xml` side-car files are used to set field names potentially longer than 10 characters, and field aliases.

7.73.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.73.2 Encoding

An attempt is made to read the code page setting in the .cpg file, or as a fallback in the LDID/codepage setting from the .dbf file, and use it to translate string fields to UTF-8 on read, and back when writing. LDID "87 / 0x57" is treated as ISO-8859-1 which may not be appropriate. The `SHAPE_ENCODING` configuration option may be used to override the encoding interpretation of the shapefile with any encoding supported by CPLRecode or to "" to avoid any recoding.

Starting with GDAL 3.1, the following metadata items are available in the "SHAPEFILE" domain:

- **LDID_VALUE**=integer: Raw LDID value from the DBF header. Only present if this value is not zero.
- **ENCODING_FROM_LDID**=string: Encoding name deduced from LDID_VALUE. Only present if LDID_VALUE is present
- **CPG_VALUE**=string: Content of the .cpg file. Only present if the file exists.
- **ENCODING_FROM_CPG**=string: Encoding name deduced from CPG_VALUE. Only present if CPG_VALUE is present
- **SOURCE_ENCODING**=string: Encoding used by GDAL to encode/recode strings. If the user has provided the `SHAPE_ENCODING` configuration option or `ENCODING` open option have been provided (included to empty value), then their value is used to fill this metadata item. Otherwise it is equal to ENCODING_FROM_CPG if it is present. Otherwise it is equal to ENCODING_FROM_LDID.

7.73.3 Spatial and attribute indexing

The OGR Shapefile driver supports spatial indexing and a limited form of attribute indexing.

The spatial indexing uses the same .qix quadtree spatial index files that are used by UMN MapServer. Spatial indexing can accelerate spatially filtered passes through large datasets to pick out a small area quite dramatically.

It can also use the ESRI spatial index files (.sbn / .sbx), but writing them is not supported currently.

To create a spatial index (in .qix format), issue a SQL command of the form

```
CREATE SPATIAL INDEX ON tablename [DEPTH N]
```

where optional DEPTH specifier can be used to control number of index tree levels generated. If DEPTH is omitted, tree depth is estimated on basis of number of features in a shapefile and its value ranges from 1 to 12.

To delete a spatial index issue a command of the form

```
DROP SPATIAL INDEX ON tablename
```

Otherwise, the MapServer shptree utility can be used:

```
shptree <shpfile> [<depth>] [<index_format>]
```

More information is available about this utility at the MapServer shptree page

Currently the OGR Shapefile driver only supports attribute indexes for looking up specific values in a unique key column. To create an attribute index for a column issue an SQL command of the form "CREATE INDEX ON tablename USING fieldname". To drop the attribute indexes issue a command of the form "DROP INDEX ON tablename". The attribute index will accelerate WHERE clause searches of the form "fieldname = value". The attribute index is actually stored as a mapinfo format index and is not compatible with any other shapefile applications.

7.73.4 Creation Issues

The Shapefile driver treats a directory as a dataset, and each Shapefile set (.shp, .shx, and .dbf) as a layer. The dataset name will be treated as a directory name. If the directory already exists it is used and existing files in the directory are ignored. If the directory does not exist it will be created.

As a special case attempts to create a new dataset with the extension .shp will result in a single file set being created instead of a directory.

ESRI shapefiles can only store one kind of geometry per layer (shapefile). On creation this is may be set based on the source file (if a uniform geometry type is known from the source driver), or it may be set directly by the user with the layer creation option SHPT (shown below). If not set the layer creation will fail. If geometries of incompatible types are written to the layer, the output will be terminated with an error.

Note that this can make it very difficult to translate a mixed geometry layer from another format into Shapefile format using ogr2ogr, since ogr2ogr has no support for separating out geometries from a source layer.

Shapefile feature attributes are stored in an associated .dbf file, and so attributes suffer a number of limitations:

- Attribute names can only be up to 10 characters long. The OGR Shapefile driver tries to generate unique field names. Successive duplicate field names, including those created by truncation to 10 characters, will be truncated to 8 characters and appended with a serial number from 1 to 99.

  For example:

  - a → a, a → a_1, A → A_2;
  - abcdefghijk → abcdefghij, abcdefghijkl → abcdefgh_1
- Only Integer, Integer64, Real, String and Date (not DateTime, just year/month/day) field types are supported. The various list, and binary field types cannot be created.
- The field width and precision are directly used to establish storage size in the .dbf file. This means that strings longer than the field width, or numbers that don't fit into the indicated field format will suffer truncation.
- Integer fields without an explicit width are treated as width 9, and extended to 10 or 11 if needed.
- Integer64 fields without an explicit width are treated as width 18, and extended to 19 or 20 if needed.
- Real (floating point) fields without an explicit width are treated as width 24 with 15 decimal places of precision.
- String fields without an assigned width are treated as 80 characters.

Also, .dbf files are required to have at least one field. If none are created by the application an "FID" field will be automatically created and populated with the record number.

The OGR shapefile driver supports rewriting existing shapes in a shapefile as well as deleting shapes. Deleted shapes are marked for deletion in the .dbf file, and then ignored by OGR. To actually remove them permanently (resulting in renumbering of FIDs) invoke the SQL 'REPACK <tablename>' via the datasource ExecuteSQL() method.

REPACK will also result in .shp being rewritten if a feature geometry has been modified with SetFeature() and resulted in a change of the size the binary encoding of the geometry in the .shp file.

REPACK is also done automatically at file closing, or at FlushCache()/SyncToDisk() time, since shapefiles with holes can cause interoperability issues with other software.

7.73.5 Field sizes

The driver knows to auto-extend string and integer fields (up to the 255 bytes limit imposed by the DBF format) to dynamically accommodate for the length of the data to be inserted.

It is also possible to force a resize of the fields to the optimal width by issuing a SQL 'RESIZE <tablename>' via the datasource ExecuteSQL() method. This is convenient in situations where the default column width (80 characters for a string field) is bigger than necessary.

7.73.6 Spatial extent

Shapefiles store the layer spatial extent in the .SHP file. The layer spatial extent is automatically updated when inserting a new feature in a shapefile. However when updating an existing feature, if its previous shape was touching the bounding box of the layer extent but the updated shape does not touch the new extent, the computed extent will not be correct. It is then necessary to force a recomputation by invoking the SQL 'RECOMPUTE EXTENT ON <tablename>' via the datasource ExecuteSQL() method. The same applies for the deletion of a shape.

7.73.7 Size Issues

Geometry: The Shapefile format explicitly uses 32bit offsets and so cannot go over 8GB (it actually uses 32bit offsets to 16bit words), but the OGR shapefile implementation has a limitation to 4GB.

Attributes: The dbf format does not have any offsets in it, so it can be arbitrarily large.

However, for compatibility with other software implementation, it is not recommended to use a file size over 2GB for both .SHP and .DBF files.

The 2GB_LIMIT=YES layer creation option can be used to strictly enforce that limit. For update mode, the *SHAPE_2GB_LIMIT* configuration option can be set to YES for similar effect. If nothing is set, a warning will be emitted when the 2GB limit is reached.

7.73.8 Compressed files

Starting with GDAL 3.1, the driver can also support reading, creating and editing .shz files (ZIP files containing the .shp, .shx, .dbf and other side-car files of a single layer) and .shp.zip files (ZIP files contains one or several layers). Creation and editing involves the creation of temporary files.

7.73.9 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are supported:

- **ENCODING=[<encoding_name>/""]:** Override the encoding interpretation of the shapefile with any encoding supported by CPLRecode or to "" to avoid any recoding.
- **DBF_DATE_LAST_UPDATE=YYYY-MM-DD:** Modification date to write in DBF header with year-month-day format. If not specified, current date is used.
- **ADJUST_TYPE=[YES/NO]:** Defaults to `NO`. Set to YES to read the whole .dbf to adjust Real->Integer/Integer64 or Integer64->Integer field types when possible. This can be used when field widths are ambiguous and that by default OGR would select the larger data type. For example, a numeric column with 0 decimal figures and with width of 10/11 character may hold Integer or Integer64, and with width 19/20 may hold Integer64 or larger integer (hold as Real)
- **ADJUST_GEOM_TYPE=[NO/FIRST_SHAPE/ALL_SHAPES]:** Defaults to `FIRST_SHAPE`. Defines how layer geometry type is computed, in particular to distinguish shapefiles that have shapes with significant values in the M dimension from the ones where the M values are set to the nodata value. By default (FIRST_SHAPE), the driver will look at the first shape and if it has M values it will expose the layer as having a M dimension.

By specifying ALL_SHAPES, the driver will iterate over features until a shape with a valid M value is found to decide the appropriate layer type.

- **AUTO_REPACK=[YES/NO]:** Defaults to `YES`. Whether the shapefile should be automatically repacked when needed, at dataset closing or at FlushCache()/SyncToDisk() time.
- **DBF_EOF_CHAR=[YES/NO]:** Defaults to `YES`. Whether the .DBF should be terminated by a 0x1A end-of-file character, as in the DBF spec and done by other software vendors. Previous GDAL versions did not write one.

7.73.10 Dataset creation options

None

7.73.11 Layer creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALDatasetCreateLayer()`* (C) or *`Dataset.CreateLayer`* (Python). The following layer creation options are supported:

- **SHPT=<type>:** Override the type of shapefile created. Can be one of NULL for a simple .dbf file with no .shp file, POINT, ARC, POLYGON or MULTIPOINT for 2D; POINTZ, ARCZ, POLYGONZ, MULTIPOINTZ or MULTIPATCH for 3D; POINTM, ARCM, POLYGONM or MULTIPOINTM for measured geometries; and POINTZM, ARCZM, POLYGONZM or MULTIPOINTZM for 3D measured geometries.
- **ENCODING=value:** Set the encoding value in the DBF file. The default value is "LDID/87". It is not clear what other values may be appropriate.
- **RESIZE=[YES/NO]:** Defaults to `NO`. Set to YES to resize fields to their optimal size. See above "Field sizes" section.
- **2GB_LIMIT=[YES/NO]:** Defaults to `NO`. Set to YES to enforce the 2GB file size for .SHP or .DBF files.
- **SPATIAL_INDEX=[YES/NO]:** Defaults to `NO`. Set to YES to create a spatial index (.qix).
- **DBF_DATE_LAST_UPDATE=<YYYY-MM-DD>:** Modification date to write in DBF header with year-month-day format. If not specified, current date is used. Note: behavior of past GDAL releases was to write 1995-07-26
- **AUTO_REPACK=[YES/NO]:** Defaults to `YES`. Whether the shapefile should be automatically repacked when needed, at dataset closing or at FlushCache()/SyncToDisk() time.
- **DBF_EOF_CHAR=[YES/NO]:** Defaults to `YES`. Whether the .DBF should be terminated by a 0x1A end-of-file character, as in the DBF spec and done by other software vendors. Previous GDAL versions did not write one.

7.73.12 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The following configuration options are available:

- **SHAPE_REWIND_ON_WRITE=[YES/NO]:** can be set to NO to prevent the shapefile writer to correct the winding order of exterior/interior rings to be conformant with the one mandated by the Shapefile specification. This can be useful in some situations where a MultiPolygon passed to the shapefile writer is not really a compliant Single Feature polygon, but originates from example from a MultiPatch object (from a Shapefile/FileGDB/PGeo datasource). Starting with GDAL 3.7, for Polygon/MultiPolygon, the default value is NO, with the effect that the winding order of rings will be determined from the outer/inner rings of the input Polygon/MultiPolygon, and not as a post process topological analysis like done in previous GDAL versions, which could cause troubles for non-planar 3D geometries.

- **SHAPE_RESTORE_SHX=[YES/NO]:** Defaults to `NO`. can be set to YES to restore broken or absent .shx file from associated .shp file during opening.
- **SHAPE_2GB_LIMIT=YES:** can be set to YES to strictly enforce the 2 GB file size limit when updating a shapefile. If nothing is set, a warning will be emitted when the 2 GB limit is reached.
- **SHAPE_ENCODING=value:** may be used to override the encoding interpretation of the shapefile with any encoding supported by *CPLRecode()* or to "" to avoid any recoding.

7.73.13 Examples

7.73.13.1 Example 1: Merging two shapefiles 'file1.shp' and 'file2.shp'

A new 'file_merged.shp' is created like this:

```
ogr2ogr file_merged.shp file1.shp
ogr2ogr -update -append file_merged.shp file2.shp -nln file_merged
```

The second command is opening file_merged.shp in update mode, and trying to find existing layers and append the features being copied.

The -nln option sets the name of the layer to be copied to.

7.73.13.2 Example 2: Building a spatial index

```
ogrinfo file1.shp -sql "CREATE SPATIAL INDEX ON file1"
```

7.73.13.3 Example 3: Resizing columns of a DBF file to their optimal size

```
ogrinfo file1.dbf -sql "RESIZE file1"
```

7.73.14 See Also

- Shapelib Page

7.74 Norwegian SOSI Standard

Driver short name

SOSI

Build dependencies

FYBA library

This driver requires the FYBA library.

7.74.1 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). Starting with GDAL 3.1, the following open options can be specified:

- **appendFieldsMap=value:** (GDAL >= 3.1) Defaults to `<empty>`. Default is that all rows for equal field names will be appended in a feature, but with this parameter you select what field this should be valid for.

7.74.1.1 Examples

Example 1

This example will convert a sosi file to a shape a file where all duplicate fields in a feature will be appended with a comma between.

```
ogr2ogr -t_srs EPSG:4258 test_poly.shp test_duplicate_fields.sos polygons
```

Example 2

This example will convert a sosi file to a shape a file where only duplicates for BEITEBRUKERID and OPPHAV will appended with a comma between.

```
ogr2ogr -t_srs EPSG:4258  test_poly.shp test_duplicate_fields.sos polygons -oo␣
↪appendFieldsMap="BEITEBRUKERID&OPPHAV"
```

Example 3

This example will convert a sosi file to a shape a file where for BEITEBRUKERID and OPPHAV will be appended with a semicolon and comma between

```
ogr2ogr -t_srs EPSG:4258  test_poly.shp test_duplicate_fields.sos polygons -oo␣
↪appendFieldsMap="BEITEBRUKERID:;&OPPHAV:,"
```

7.75 SQLite / Spatialite RDBMS

Driver short name

SQLite

Build dependencies

libsqlite3 or libspatialite

OGR optionally supports spatial and non-spatial tables stored in SQLite 3.x database files. SQLite is a "light weight" single file based RDBMS engine with fairly complete SQL semantics and respectable performance.

The driver can handle "regular" SQLite databases, as well as Spatialite databases (spatial enabled SQLite databases). The type of an existing database can be checked from the SQLITE debug info value "OGR style SQLite DB found/ SpatiaLite DB found/SpatiaLite v4 DB found" obtained by running `ogrinfo db.sqlite --debug on`

The SQLite driver can also read databases with *RasterLite2 raster coverages*.

The SQLite database is essentially typeless, but the SQLite driver will attempt to classify attributes field as text, integer or floating point based on the contents of the first record in a table. Datetime field types are also handled.

The "JSonStringList", "JSonIntegerList", "JSonInteger64List" and "JSonRealList" SQLite declaration types are used to map the corresponding OGR StringList, IntegerList, Integer64List and RealList types. The field values are then encoded as JSON arrays, with proper CSV escaping.

SQLite databases often do not work well over NFS, or some other networked file system protocols due to the poor support for locking. It is safest to operate only on SQLite files on a physical disk of the local system.

SQLite is an optionally compiled in driver. It is not compiled in by default.

By default, SQL statements are passed directly to the SQLite database engine. It's also possible to request the driver to handle SQL commands with *OGR SQL* engine, by passing **"OGRSQL"** string to the ExecuteSQL() method, as name of the SQL dialect.

The *OGR_SQLITE_SYNCHRONOUS* configuration option has been added. When set to OFF, this issues a 'PRAGMA synchronous = OFF' command to the SQLite database. This has the advantage of speeding-up some write operations (e.g. on EXT4 filesystems), but at the expense of data safety w.r.t system/OS crashes. So use it carefully in production environments and read the SQLite related documentation.

Any SQLite pragma can be specified with the *OGR_SQLITE_PRAGMA* configuration option. The syntax is `OGR_SQLITE_PRAGMA = "pragma_name=pragma_value[,pragma_name2=pragma_value2]*"`.

7.75.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.75.2 "Regular" SQLite databases

The driver looks for a geometry_columns table laid out as defined loosely according to OGC Simple Features standards, particularly as defined in FDO RFC 16. If found it is used to map tables to layers.

If geometry_columns is not found, each table is treated as a layer. Layers with a WKT_GEOMETRY field will be treated as spatial tables, and the WKT_GEOMETRY column will be read as Well Known Text geometry.

If geometry_columns is found, it will be used to lookup spatial reference systems in the spatial_ref_sys table.

While the SQLite driver supports reading spatial data from records, there is no support for spatial indexing, so spatial queries will tend to be slow (use Spatialite for that). Attributes queries may be fast, especially if indexes are built for appropriate attribute columns using the "CREATE INDEX ON ( )" SQL command.

The driver also supports reading and writing the following non-linear geometry types :CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE and MULTISURFACE. Note: this is not true for Spatialite databases, since those geometry types are not supported by current Spatialite versions.

7.75.3 Tables with multiple geometry columns

Layers with multiple geometry columns can be created, modified or read, following new API described in rfc-41

7.75.4 REGEXP operator

By default, the REGEXP operator has no implementation in SQLite. With OGRbuilt against the PCRE library, the REGEXP operator is available in SQL statements run by OGR.

7.75.5 Using the SpatiaLite library (Spatial extension for SQLite)

The SQLite driver can read and write SpatiaLite databases. Creating or updating a spatialite database requires explicit linking against SpatiaLite library (version >= 2.3.1). Explicit linking against SpatiaLite library also provides access to functions provided by this library, such as spatial indexes, spatial functions, etc...

A few examples :

```
# Duplicate the sample database provided with SpatiaLite
ogr2ogr -f SQLite testspatialite.sqlite test-2.3.sqlite  -dsco SPATIALITE=YES

# Make a request with a spatial filter. Will work faster if spatial index has
# been created and explicit linking against SpatiaLite library.
ogrinfo testspatialite.sqlite Towns -spat 754000 4692000 770000 4924000
```

7.75.6 Opening with 'VirtualShape:'

(Require Spatialite support)

It is possible to open on-the-fly a shapefile as a VirtualShape with Spatialite. The syntax to use for the datasource is "VirtualShape:/path/to/shapefile.shp" (the shapefile must be a "real" file).

This gives the capability to use the spatial operations of Spatialite (note that spatial indexes on virtual tables are not available).

7.75.7 The SQLite SQL dialect

The SQLite SQL engine can be used to run SQL queries on any OGR datasource if using the *SQL SQLite dialect*.

7.75.8 The VirtualOGR SQLite extension

The GDAL/OGR library can be loaded as a SQLite extension. The extension is loaded with the load_extension(gdal_library_name) SQL function, where gdal_library_name is typically libgdal.so on Unix/Linux, gdal110.dll on Windows, etc..

After the extension is loaded, a virtual table, corresponding to a OGR layer, can be created with one of the following SQL statement :

```
CREATE VIRTUAL TABLE table_name USING VirtualOGR(datasource_name);
CREATE VIRTUAL TABLE table_name USING VirtualOGR(datasource_name, update_mode);
CREATE VIRTUAL TABLE table_name USING VirtualOGR(datasource_name, update_mode, layer_
↪name);
CREATE VIRTUAL TABLE table_name USING VirtualOGR(datasource_name, update_mode, layer_
↪name, expose_ogr_style);
```

where :

- *datasource_name* is the connection string to any OGR datasource.

- *update_mode* = 0 for read-only mode (default value) or 1 for update mode.
- *layer_name* = the name of a layer of the opened datasource.
- *expose_ogr_style* = 0 to prevent the OGR_STYLE special from being displayed (default value) or 1 to expose it.

Note: *layer_name* does not need to be specified if the datasource has only one single layer.

From the sqlite3 console, a typical use case is :

```
sqlite> SELECT load_extension('libgdal.so');

sqlite> SELECT load_extension('mod_spatialite.so');

sqlite> CREATE VIRTUAL TABLE poly USING VirtualOGR('poly.shp');

sqlite> SELECT *, ST_Area(GEOMETRY) FROM POLY;
215229.266|168.0|35043411||215229.265625
247328.172|179.0|35043423||247328.171875
261752.781|171.0|35043414||261752.78125
547597.188|173.0|35043416||547597.2109375
15775.758|172.0|35043415||15775.7578125
101429.977|169.0|35043412||101429.9765625
268597.625|166.0|35043409||268597.625
1634833.375|158.0|35043369||1634833.390625
596610.313|165.0|35043408||596610.3359375
5268.813|170.0|35043413||5268.8125
```

Alternatively, you can use the *ogr_datasource_load_layers(datasource_name[, update_mode[, prefix]])* function to automatically load all the layers of a datasource.

```
sqlite> SELECT load_extension('libgdal.so');

sqlite> SELECT load_extension('mod_spatialite.so');

sqlite> SELECT ogr_datasource_load_layers('poly.shp');
1
sqlite> SELECT * FROM sqlite_master;
table|poly|poly|0|CREATE VIRTUAL TABLE "poly" USING VirtualOGR('poly.shp', 0, 'poly')
```

Refer to the *SQL SQLite dialect* for an overview of the capabilities of VirtualOGR tables.

7.75.9 Creation Issues

The SQLite driver supports creating new SQLite database files, or adding tables to existing ones.

7.75.9.1 Transaction support

The driver implements transactions at the database level, per rfc-54

7.75.10 Relationships

Added in version 3.6.

Relationship retrieval is supported. Relationships will be reported for tables which utilize FOREIGN KEY constraints.

Relationship creation is supported since GDAL 3.7, for one-to-many relationships only.

7.75.10.1 Dataset open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are supported:

- **LIST_ALL_TABLES=[YES/NO]:** This may be "YES" to force all tables, including non-spatial ones, to be listed.
- **LIST_VIRTUAL_OGR=[YES/NO]:** This may be "YES" to force VirtualOGR virtual tables to be listed. This should only be enabled on trusted datasources to avoid potential safety issues.
- **PRELUDE_STATEMENTS=value:** (GDAL >= 3.2) SQL statement(s) to send on the SQLite3 connection before any other ones. In case of several statements, they must be separated with the semi-column (;) sign. This option may be useful to attach another database to the current one and issue cross-database requests.

  > **Note**
  >
  > The other database must be of a type recognized by this driver, so its geometry blobs are properly recognized (so typically not a GeoPackage one)

- **OGR_SCHEMA=<filename>|<json string>:** (GDAL >= 3.11.0) Partially or totally overrides the auto-detected schema to use for creating the layer. The overrides are defined as a JSON list of field definitions. This can be a filename, a URL or JSON string conformant with the ogr_fields_override.schema.json schema

7.75.10.2 Database creation options

Dataset creation options can be specified in command-line tools using the syntax `-dsco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following dataset creation options are supported:

- **METADATA=[YES/NO]:** This can be used to avoid creating the geometry_columns and spatial_ref_sys tables in a new database. By default these metadata tables are created when a new database is created.
- **SPATIALITE=[YES/NO]:** Create the SpatiaLite flavor of the metadata tables, which are a bit differ from the metadata used by this OGR driver and from OGC specifications. Implies *METADATA=YES*. Please note: OGR must be linked against *libspatialite* in order to support insert/write on SpatiaLite; if not, *read-only* mode is enforced.

  Attempting to perform any insert/write on SpatiaLite skipping the appropriate library support simply produces broken (corrupted) DB-files.

  Important notice: when the underlying *libspatialite* is v.2.3.1 (or any previous version) any Geometry will be casted to 2D [XY], because earlier versions of this library are simply able to support 2D [XY] dimensions. Version 2.4.0 (or any subsequent) is required in order to support 2.5D [XYZ].
- **INIT_WITH_EPSG=[YES/NO]:** Insert the content of the EPSG CSV files into the spatial_ref_sys table. Defaults to NO for regular SQLite databases. Please note: if *SPATIALITE=YES* and the underlying *libspatialite* is v2.4 or v3.X, *INIT_WITH_EPSG* is ignored; those library versions will unconditionally load the EPSG dataset into the spatial_ref_sys table when creating a new DB (*self-initialization*). Starting with libspatialite 4.0, *INIT_WITH_EPSG* defaults to YES, but can be set to NO.

7.75.10.3 Layer creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALDatasetCreateLayer()* (C) or *Dataset.CreateLayer* (Python). The following layer creation options are supported:

- **FORMAT=[WKB/WKT/SPATIALITE]:** Defaults to `WKB`. Controls the format used for the geometry column. By default WKB (Well Known Binary) is used. This is generally more space and processing efficient, but harder

to inspect or use in simple applications than WKT (Well Known Text). SpatiaLite extension uses its own binary format to store geometries and you can choose it as well. It will be selected automatically when SpatiaLite database is opened or created with *SPATIALITE=YES* option. SPATIALITE value is available.

- **GEOMETRY_NAME=value:** By default OGR creates new tables with the geometry column named GEOMETRY (or WKT_GEOMETRY if *FORMAT=WKT*). If you wish to use a different name, it can be supplied with the *GEOMETRY_NAME* layer creation option.
- **LAUNDER=[YES/NO]:** Defaults to `YES`. Controls whether layer and field names will be laundered for easier use in SQLite. Laundered names will be converted to lower case and some special characters(' - #) will be changed to underscores.
- **SPATIAL_INDEX=[YES/NO]:** Defaults to `YES`. If the database is of the SpatiaLite flavor, and if OGR is linked against libspatialite, this option can be used to control if a spatial index must be created.
- **COMPRESS_GEOM=[YES/NO]:** Defaults to `NO`. If the format of the geometry BLOB is of the SpatiaLite flavor, this option can be used to control if the compressed format for geometries (LINESTRINGs, POLYGONs) must be used. This format is understood by Spatialite v2.4 (or any subsequent version). Default to NO. Note: when updating an existing Spatialite DB, the *COMPRESS_GEOM* configuration option can be set to produce similar results for appended/overwritten features.
- **SRID=value:** Used to force the SRID number of the SRS associated with the layer. When this option isn't specified and that a SRS is associated with the layer, a search is made in the spatial_ref_sys to find a match for the SRS, and, if there is no match, a new entry is inserted for the SRS in the spatial_ref_sys table. When the SRID option is specified, this search (and the eventual insertion of a new entry) will not be done : the specified SRID is used as such.
- **COMPRESS_COLUMNS=<column_name1[,column_name2, ...]>:** A list of (String) columns that must be compressed with ZLib DEFLATE algorithm. This might be beneficial for databases that have big string blobs. However, use with care, since the value of such columns will be seen as compressed binary content with other SQLite utilities (or previous OGR versions). With OGR, when inserting, modifying or querying compressed columns, compression/decompression is done transparently. However, such columns cannot be (easily) queried with an attribute filter or WHERE clause. Note: in table definition, such columns have the "VARCHAR_deflate" declaration type.
- **FID=value:** Defaults to `OGC_FID`. Name of the FID column to create.
- **STRICT=[YES/NO]:** (GDAL >= 3.3.5) Defaults to `NO`. (SQLite >= 3.37) Whether the table should be created as a strict table, that is strong column type checking. This normally has little influence when operating only through OGR, since it has typed columns, but can help to strengthen database integrity when the database might be edited by external tools. Note that databases that contain STRICT tables can only be read by SQLite >= 3.37. The set of column data types supported in STRICT mode is: Integer, Integer64, Real, String, DateTime, Date and Time. The *COMPRESS_COLUMNS* option is ignored in strict mode.

7.75.11 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). The following configuration options are available:

- **SQLITE_LIST_ALL_TABLES=[YES/NO]:** Defaults to `NO`. Set to "YES" to list all tables (not just the tables listed in the geometry_columns table). This can also be done using the *LIST_ALL_TABLES* open option.
- **OGR_PROMOTE_TO_INTEGER64=[YES/NO]:** Defaults to `NO`. Whether to read fields with type `INTEGER` as 64-bit integers.
- **OGR_SQLITE_LIST_VIRTUAL_OGR=[YES/NO]:** Defaults to `NO`. Set to "YES" to list VirtualOGR layers. Defaults to "NO" as there might be some security implications if a user is provided with a file and doesn't know that there are virtual OGR tables in it.

- **OGR_SQLITE_JOURNAL=value:** can be used to set the journal mode of the SQLite file, see also https://www.sqlite.org/pragma.html#pragma_journal_mode.
- **OGR_SQLITE_CACHE=value:** see *Performance hints*.
- **OGR_SQLITE_SYNCHRONOUS=value:** see *Performance hints*.
- **OGR_SQLITE_LOAD_EXTENSIONS=[<extension1,...,extensionN>/ENABLE_SQL_LOAD_EXTENSION]:** (GDAL >= 3.5.0) Comma separated list of names of shared libraries containing extensions to load at database opening. If a file cannot be loaded directly, attempts are made to load with various operating-system specific extensions added. So for example, if "samplelib" cannot be loaded, then names like "samplelib.so" or "samplelib.dylib" or "samplelib.dll" might be tried also. The special value `ENABLE_SQL_LOAD_EXTENSION` can be used to enable the use of the SQL `load_extension()` function, which is normally disabled in standard builds of sqlite3. Loading extensions as a potential security impact if they are untrusted.
- **OGR_SQLITE_PRAGMA=value:** with this option any SQLite pragma can be specified. The syntax is `OGR_SQLITE_PRAGMA = "pragma_name=pragma_value[,pragma_name2=pragma_value2]*"`.
- **SQLITE_USE_OGR_VFS=[YES/NO]:** YES enables extra buffering/caching by the GDAL/OGR I/O layer and can speed up I/O. More information *here*. Be aware that no file locking will occur if this option is activated, so concurrent edits may lead to database corruption.
- **COMPRESS_GEOM=[YES/NO]:** Defaults to `NO`. Equivalent of *COMPRESS_GEOM* layer creation option for use when updating or appending to an existing layer.

7.75.12 Performance hints

SQLite is a Transactional DBMS; while many INSERT statements are executed in close sequence, BEGIN TRANSACTION and COMMIT TRANSACTION statements have to be invoked appropriately (with the OGR_L_StartTransaction() / OGR_L_CommitTransaction()) in order to get optimal performance. By default, if no transaction is explicitly started, SQLite will autocommit on every statement, which will be slow. If using ogr2ogr, its default behavior is to COMMIT a transaction every 100000 inserted rows. The **-gt** argument allows explicitly setting the number of rows for each transaction.

SQLite usually has a very minimal memory foot-print; just about 20MB of RAM are reserved to store the internal Page Cache [merely 2000 pages]. This value too may well be inappropriate under many circumstances, most notably when accessing some really huge DB-file containing many tables related to a corresponding Spatial Index. Explicitly setting a much more generously dimensioned internal Page Cache may often help to get a noticeably better performance. You can explicitly set the internal Page Cache size using the configuration option *OGR_SQLITE_CACHE value* [*value* being measured in MB]; if your HW has enough available RAM, defining a Cache size as big as 512MB (or even 1024MB) may sometimes help a lot in order to get better performance.

Setting the *OGR_SQLITE_SYNCHRONOUS* configuration option to *OFF* might also increase performance when creating SQLite databases (although at the expense of integrity in case of interruption/crash).

If many source files will be collected into the same Spatialite table, it can be much faster to initialize the table without a spatial index by using -lco *SPATIAL_INDEX=NO* and to create spatial index with a separate command after all the data are appended. Spatial index can be created with ogrinfo command

```
ogr2ogr -f SQLite -dsco SPATIALITE=YES db.sqlite first.shp -nln the_table -lco SPATIAL_
↪INDEX=NO
ogr2ogr -append db.sqlite second.shp -nln the_table
...
ogr2ogr -append db.sqlite last.shp -nln the_table
ogrinfo db.sqlite -sql "SELECT CreateSpatialIndex('the_table','GEOMETRY')"
```

If a database has gone through editing operations, it might be useful to run a VACUUM query to compact and optimize it.

```
ogrinfo db.sqlite -sql "VACUUM"
```

7.75.13 Secure deletion

Depending on how SQLite3 is built, secure deletion might or might not be enabled. Starting with GDAL 3.10, secure deletion is always enabled, unless SECURE_DELETE is specified through the *OGR_SQLITE_PRAGMA* configuration option. Note that secure deletion does not recover potential lost space, so running a VACUUM query is recommended to fully optimized a database that has been subject to updates or deletions.

7.75.14 Examples

7.75.14.1 Example 1: Converting a non-spatial SQLite table into a GeoPackage

```
ogr2ogr \
  -f "GPKG" output.gpkg \
  input.sqlite \
  -sql \
  "SELECT
     *,
     MakePoint(longitude, latitude, 4326) AS geometry
   FROM
     my_table" \
  -nln "location" \
  -s_srs "EPSG:4326"
```

7.75.14.2 Example 2: Performing a join between two SQLite/Spatialite databases

```
ogrinfo my_spatial.db \
    -sql "SELECT poly.id, other.foo FROM poly JOIN other_schema.other USING (id)" \
    -oo PRELUDE_STATEMENTS="ATTACH DATABASE 'other.db' AS other_schema"
```

7.75.15 Credits

- Development of the OGR SQLite driver was supported by DM Solutions Group and GoMOOS.
- Full support for SpatiaLite was contributed by A.Furieri, with funding from Regione Toscana

7.75.16 Links

- http://www.sqlite.org: Main SQLite page.
- https://www.gaia-gis.it/fossil/libspatialite/index: SpatiaLite extension to SQLite.
- FDO RFC 16: FDO Provider for SQLite
- *RasterLite2 driver*

7.76 Storage and eXchange Format - SXF

Driver short name

SXF

Driver built-in by default

This driver is built-in by default

This driver reads SXF files, open format often associated with Russian GIS Software Panorama.

The driver is read only, but supports deletion of data source. The driver supports SXF binary files version 3.0 and higher.

The SXF layer support the following capabilities:

- Strings as UTF8
- Random Read
- Fast Feature Count
- Fast Get Extent
- Fast Set Next By Index

The driver uses classifiers (RSC files) to map feature from SXF to layers. Features that do not belong to any layer are put to the layer named "Not_Classified". The layers with zero features are not present in data source.

To be used automatically, the RSC file should have the same name as SXF file. User can provide RSC file path using config option **SXF_RSC_FILENAME**. This config option overrides the use of same name RSC.

The RSC file usually stores long and short layer name. The long name is usually in Russian, and short in English. The **SXF_LAYER_FULLNAME** config option allows choosing which layer names to use. If SXF_LAYER_FULLNAME is TRUE - the driver uses long names, if FALSE - short.

The attributes are read from SXF file. Maximum number of fields is created for the same layer features with different number of attributes. If attribute has a code mapped to RSC file, driver adds only the code (don't get real value from RSC, as the value type may differ from field type).

If config option **SXF_SET_VERTCS** set to ON, the layers spatial reference will include vertical coordinate system description if exist.

Since GDAL 3.1 config options can be passed as driver open options.

7.76.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.76.2 See Also

- Panorama web page
- SXF binary format description v.4 (rus) - pdf
- SXF binary format description v.4 (rus) - doc

- SXF format description v.3 (rus)
- RSC format description (rus)
- Test spatial data in SXF format (rus)
- Some RSC files (rus)

7.77 U.S. Census TIGER/Line

Driver short name

TIGER

Driver built-in by default

This driver is built-in by default

TIGER/Line file sets are support for read access.

TIGER/Line files are a digital database of geographic features, such as roads, railroads, rivers, lakes, political boundaries, census statistical boundaries, etc. covering the entire United States. The data base contains information about these features such as their location in latitude and longitude, the name, the type of feature, address ranges for most streets, the geographic relationship to other features, and other related information. They are the public product created from the Census Bureau's TIGER (Topologically Integrated Geographic Encoding and Referencing) data base of geographic information. TIGER was developed at the Census Bureau to support the mapping and related geographic activities required by the decennial census and sample survey programs.

Note that the TIGER/Line product does not include census demographic statistics. Those are sold by the Census Bureau in a separate format (not directly supported by FME), but those statistics do relate back to census blocks in TIGER/Line files.

To open a TIGER/Line dataset, select the directory containing one or more sets of data files. The regions are counties, or county equivalents. Each county consists of a series of files with a common basename, and different extensions. For instance, county 1 in state 26 (Michigan) consists of the following file set in Tiger98.

```
TGR26001.RT1
TGR26001.RT2
TGR26001.RT3
TGR26001.RT4
TGR26001.RT5
TGR26001.RT6
TGR26001.RT7
TGR26001.RT8
TGR26001.RT9
TGR26001.RTA
TGR26001.RTC
TGR26001.RTH
TGR26001.RTI
TGR26001.RTP
TGR26001.RTR
TGR26001.RTS
TGR26001.RTZ
```

The TIGER/Line coordinate system is hardcoded to NAD83 lat/long degrees. This should be appropriate for all recent years of TIGER/Line production.

There is no update or creation support in the TIGER/Line driver.

The reader was implemented for TIGER/Line 1998 files, but some effort has gone into ensuring compatibility with 1992, 1995, 1997, 1999, 2000, 2002, 2003 and 2004 TIGER/Line products as well. The 2005 products have also been reported to work fine. It is believe that any TIGER/Line product from the 1990's should work with the reader, with the possible loss of some version specific information.

7.77.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.77.2 Feature Representation

With a few exceptions, a feature is created for each record of a TIGER/Line data file. Each file (i.e. .RT1, .RTA) is translated to an appropriate OGR feature type, with attribute names matching those in the TIGER/Line product manual.

The TIGER/Line RT (record type), and VERSION attributes are generally discarded, but the MODULE attribute is added to each feature. The MODULE attribute contains the basename (eg. TGR26001) of the county module from which the feature originated. For some keys (such as LAND, POLYID, and CENID) this MODULE attribute is needed to make the key unique when the dataset (directory) consists of more than one county of data.

Following is a list of feature types, and their relationship to the TIGER/Line product.

7.77.2.1 CompleteChain

A CompleteChain is a polyline with an associated TLID (TIGER/Line ID). The CompleteChain features are established from a type 1 record (Complete Chain Basic Data Record), and if available it is associated type 3 record (Complete Chain Geographic Entity Codes). As well, any type 2 records (Complete Chain Shape Coordinates) available are used to fill in intermediate shape points on the arc. The TLID is the primary key, and is unique within the entire national TIGER/Line coverage.

These features always have a line geometry.

7.77.2.2 AltName

These features are derived from the type 4 record (Index to Alternate Feature Identifiers), and relate a TLID to 1 to 4 alternate feature name numbers (the FEAT attribute) which are kept separately as FeatureIds features. The standard reader pipeline attaches the names from the FeatureIds features as array attributes ALT_FEDIRS{}, ALT_FEDIRP{}, ALT_FENAME{} and ALT_FETYPE{}. The ALT_FENAME{} is a list of feature names associated with the TLID on the AltName feature.

Note that zero, one or more AltName records may exist for a given TLID, and each AltName record can contain between one and four alternate names. Because it is still very difficult to utilize AltName features to relate alternate names to CompleteChains, it is anticipated that the standard reader pipeline for TIGER/Line files will be upgraded in the future, resulting in a simplification of alternate names.

These features have no associated geometry.

7.77.2.3 FeatureIds

These features are derived from type 5 (Complete Chain Feature Identifiers) records. Each feature contains a feature name (FENAME), and it is associated feature id code (FEAT). The FEAT attribute is the primary key, and is unique within the county module. FeatureIds have a one to many relationship with AltName features, and KeyFeatures features.

These features have no associated geometry.

7.77.2.4 ZipCodes

These features are derived from type 6 (Additional Address Range and ZIP Code Data) records. These features are intended to augment the ZIP Code information kept directly on CompleteChain features, and there is a many to one relationship between ZipCodes features and CompleteChain features.

These features have no associated geometry.

7.77.2.5 Landmarks

These features are derived from type 7 (Landmark Features) records. They relate to point, or area landmarks. For area landmarks there is a one to one relationship with an AreaLandmark record. The LAND attribute is the primary key, and is unique within the county module.

These features may have an associated point geometry. Landmarks associated with polygons will not have the polygon geometry attached. It would need to be collected (via the AreaLandmark feature) from a Polygon feature.

7.77.2.6 AreaLandmarks

These features are derived from type 8 (Polygons Linked to Area Landmarks) records. Each associates a Landmark feature (attribute LAND) with a Polygon feature (attribute POLYID). This feature has a many to many relationship with Polygon features.

These features have no associated geometry.

7.77.2.7 KeyFeatures

These features are derived from type 9 (Polygon Geographic Entity Codes) records. They may be associated with a FeatureIds feature (via the FEAT attribute), and a Polygon feature (via the POLYID attribute).

These features have no associated geometry.

7.77.2.8 Polygon

These features are derived from type A (Polygon Geographic Entity Codes) records and if available the related type S (Polygon Additional Geographic Entity Codes) records. The POLYID attribute is the primary key, uniquely identifying a polygon within a county module.

These features do not have any geometry associated with them as read by the OGR TIGER driver. It needs to be externally related using the PolyChainLink. The gdal/pymod/samples/tigerpoly.py script may be used to read a TIGER dataset and extract the polygon layer **with geometry** as a shapefile.

7.77.2.9 EntityNames

These features are derived from type C (Geographic Entity Names) records.

These features have no associated geometry.

7.77.2.10 IDHistory

These features are derived from type H (TIGER/Line ID History) records. They can be used to trace the splitting, merging, creation and deletion of CompleteChain features.

These features have no associated geometry.

7.77.2.11 PolyChainLink

These features are derived from type I (Link Between Complete Chains and Polygons) records. They are normally all consumed by the standard reader pipeline while attaching CompleteChain geometries to Polygon features to establish their polygon geometries. PolyChainLink features have a many to one relationship with Polygon features, and a one to one relationship with CompleteChain features.

These features have no associated geometry.

7.77.2.12 PIP

These features are derived from type P (Polygon Internal Point) records. They relate to a Polygon feature via the POLYID attribute, and can be used to establish an internal point for Polygon features.

These features have a point geometry.

7.77.2.13 ZipPlus4

These features are derived from type Z (ZIP+4 Codes) records. ZipPlus4 features have a many to one relationship with CompleteChain features.

These features have no associated geometry.

7.77.3 See Also

http://www.census.gov/geo/www/tiger/: More information on the TIGER/Line file format, and data product can be found on this U.S. Census web page.

7.78 TileDB -- TileDB vector

Driver short name

TileDB

Added in version 3.7.

Build dependencies

TileDB >= 2.7

GDAL can read and write TileDB sparse arrays containing vector data.

The driver relies on the Open Source TileDB library (MIT licensed).

7.78.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.78.2 Supported datasets

The driver can read TileDB sparse arrays that contain at least 2 dimensions of type Float64. By default dimensions names `_X` and `_Y` are looked up, but this can be customized with the *DIM_X* and *DIM_Y* open options.

Attributes (or extra dimensions) of the following TileDB data types are recognized as OGR fields: Bool, Int16, Int32, Int64, Float32, Float64, String_ASCII, String_UTF8, Blob, varying size UInt8 (as Binary), DateTime_Day, DateTime_MS, Time_MS. Attributes of other data types will be ignored.

If a `wkb_geometry` named attribute of type Binary is found, it is used as the geometry column, and is assumed to contain WKB encoded geometries. If there is no such column, Point geometries are assumed and the X, Y (and optional Z) dimensions are used to create the corresponding OGR point geometries.

7.78.3 Dataset connection string

Valid dataset connection strings are paths to a local or remote TileDB sparse array, or to a local or remote TileDB group that contains TileDB sparse arrays.

7.78.4 Filtering

The bounding box of OGR spatial filters is forwarded to the TileDB query engine.

OGR attribute filters are translated to TileDB query conditions for the following elements:

- AND
- OR
- NOT
- comparisons of the form "attribute_name operator constant" where operator is =, <>, <, <=, >, >=
- attribute_name IS NULL
- attribute_name IS NOT NULL
- attribute_name IN (value1, ... valueN)
- attribute_name BETWEEN min_val AND max_val

Other OGR attribute filter elements may be used, but this may cause the filter to be fully or partially evaluated on OGR side. For example, given the filter "int_attribute = 5 AND string_attribute LIKE '%foo%'", the condition on int_attribute will be translated as a TileDB query condition, and the right side of the AND operation will be evaluated as a post-filter on OGR side.

7.78.5 Arrow C Stream data interface

The driver has an efficient implementation of the *Arrow C Stream data interface*

The `INCLUDE_FID=YES/NO` and `MAX_FEATURES_IN_BATCH=number` options of *`OGRLayer::GetArrowStream()`* are supported. If MAX_FEATURES_IN_BATCH is not specified, it defaults to the value of the BATCH_SIZE open option.

7.78.6 Creation issues

By default, when creating a layer, the driver will create a TileDB sparse array at the location of the dataset connection string. Consequently only one OGR layer can be created in that mode. If several layers need to be created and accessible through a OGR dataset connection, a TileDB group needs to be created to point to the different arrays (layers), by specifying the *`CREATE_GROUP=YES`* dataset creation option.

The driver supports appending features to exiting layers.

The driver does not support adding new fields to a layer where features have already been written.

7.78.7 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options are available:

- **TILEDB_CONFIG=<filename>:** A local file with TileDB configuration options
- **TILEDB_TIMESTAMP=<integer>:** Open array at this timestamp. The timestamp should be greater than 0.
- **BATCH_SIZE=<integer>:** Defaults to `500000`. Number of features to fetch/write at once.
- **DIM_X=value:** Defaults to `_X`. Name of the X dimension.
- **DIM_Y=value:** Defaults to `_Y`. Name of the Y dimension.
- **DIM_Z=value:** Defaults to `_Z`. Name of the Z dimension.

7.78.8 Dataset creation options

Dataset creation options can be specified in command-line tools using the syntax `-dsco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALCreate()`* (C) or *`Driver.Create`* (Python). The following dataset creation options are available:

- **TILEDB_CONFIG=<filename>:** A local file with TileDB configuration options
- **CREATE_GROUP=[YES/NO]:** Defaults to `NO`. (TileDB >= 2.9) Whether to create a group for multiple layer support. When set to YES, a TileDB group will be created in the directory of the dataset name, and layers will be created as members of the group, and written in subdirectories of a `layers` subdirectory.

7.78.9 Layer creation options

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALDatasetCreateLayer()`* (C) or *`Dataset.CreateLayer`* (Python). The following layer creation options are available:

- **COMPRESSION=[NONE/GZIP/ZSTD/LZ4/RLE/BZIP2/DOUBLE-DELTA/POSITIVE_DELTA]:** Defaults to `NONE`. compression method for dimensions and attributes.
- **COMPRESSION_LEVEL=<integer>:** compression level
- **BATCH_SIZE=<integer>:** Defaults to `500000`. Number of features to write at once.
- **TILE_CAPACITY=<integer>:** Defaults to `10000`. Number of non-empty cells stored in a data tile.

- **TILE_EXTENT=<float>:** The square TileDB tile extents in the X and Y dimensions. Default is auto-calculated.
- **TILE_Z_EXTENT=<float>:** The TIleDB tile extent in the Z dimension. Default is auto-calculated.
- **BOUNDS=<minx,miny,[minz,]maxx,maxy[, maxz]>:** Specify bounds for sparse array. If not specified, the CRS passed at layer creation will be used to infer default values for bounds.
- **ADD_Z_DIM=[AUTO/YES/NO]:** Defaults to `AUTO`. Whether to add a Z dimension. In the default AUTO mode, a Z dimension is only added if the layer geometry type has a Z component or is unknown. Setting it to YES or NO explicitly force or disable creation of a Z dimension.
- **FID=value:** Defaults to `FID`. Feature id column name. Set to empty to disable its creation.
- **GEOMETRY_NAME=value:** Defaults to `wkb_geometry`. Name of the geometry column that will receive WKB encoded geometries. Set to empty to disable its creation (only for point).
- **TILEDB_TIMESTAMP=<integer>:** Timestamp at which to create the array. The timestamp should be greater than 0.
- **TILEDB_STRING_TYPE=[UTF8/ASCII]:** Which TileDB type to create string attributes. Default is UTF8 starting with TileDB 2.14 (ASCII for earlier versions)

7.78.10 See Also

- TileDB home page
- *TileDB raster* documentation page

7.79 TopoJSON driver

Driver short name

TopoJSON

Driver built-in by default

This driver is built-in by default

The driver can read the TopoJSON format. The driver does not support writing TopoJSON datasets.

7.79.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.79.2 Datasource

The driver accepts three types of sources of data:

- Uniform Resource Locator (URL) - a Web address to perform HTTP request.
- Plain text file with TopoJSON data - identified from the file extension .json or .topojson
- Text passed directly and encoded in Topo JSON

The URL/filename/text might be prefixed with TopoJSON: to avoid any ambiguity with other drivers. Alternatively, starting with GDAL 3.10, specifying the `-if TopoJSON` option to command line utilities accepting it, or `TopoJSON` as the only value of the `papszAllowedDrivers` of *GDALOpenEx()*, also forces the driver to recognize the passed URL/filename/text.

7.79.3 Examples

7.79.3.1 Example 1: Reading a TopoJSON file with multiple layers

```
gdal vector info "https://cdn.jsdelivr.net/npm/us-atlas@3/counties-albers-10m.json"
```

7.79.3.2 Example 2: Writing a single layer from a TopoJSON file to GeoJSON file

```
gdal vector convert "https://cdn.jsdelivr.net/npm/us-atlas@3/counties-albers-10m.json"␣
↪counties-albers-10m.geojson --layer counties
```

7.79.4 See Also

- *GeoJSON driver*
- TopoJSON Format Specification

7.80 VDV - VDV-451/VDV-452/INTREST Data Format

Driver short name

VDV

Driver built-in by default

This driver is built-in by default

This driver can read and create text files following the VDV-451 file format, which is a text format similar to CSV files, potentially containing several layers within the same file.

It supports in particular reading 2 "profiles" :

- (read/write) VDV-452 standard for route network / timetable
- (read/only) "INTREST Data format" used by the Austrian official open government street graph

The generic reader/writer for VDV-451/VDV-452 can support arbitrarily large files. For the INTREST data case, for combined layers in a single file, the driver ingests the whole file in memory to reconstruct the Link layer.

Interleave reading among layers is supported in files with multiple layers.

7.80.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.80.2 Creation issues

The driver can create new layers (either in the same file, or in separate files in the same directory). It can append a new layer into an existing file, but it cannot append/edit/delete features to an existing layer, or modify the attribute structure of an existing layer after features have been written.

Dataset creation options can be specified in command-line tools using the syntax `-dsco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALCreate()* (C) or *Driver.Create* (Python). The following dataset creation options are available:

- **SINGLE_FILE=[YES/NO]:** Defaults to `YES`. Whether several layers should be put in the same file. If NO, the name is assumed to be a directory name.

Layer creation options can be specified in command-line tools using the syntax `-lco <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALDatasetCreateLayer()* (C) or *Dataset.CreateLayer* (Python). The following layer creation options are available:

- **EXTENSION=value:** Defaults to `x10`. Extension used when creating files in separate layers, i.e. only for *SINGLE_FILE=NO* dataset creation option.
- **PROFILE=[GENERIC/VDV-452/VDV-452-ENGLISH/VDV-452-GERMAN]:** Defaults to `GENERIC`. Describe which profile the writer should conform to. VDV-452 will restrict layer and field names to be the one allowed by the VDV-452 standard (either in English or German). VDV-452-ENGLISH and VDV-452-GERMAN will restrict the VDV-452 to the specified language. The configuration file describing VDV-452 table and field names is vdv452.xml located in the GDAL_DATA directory.
- **PROFILE_STRICT=[YES/NO]:** Defaults to `NO`. Whether checks of profile should be strict. In strict mode, unexpected layer or field names will be rejected.
- **CREATE_ALL_FIELDS=[YES/NO]:** Defaults to `YES`. Whether all fields of predefined profiles should be created at layer creation.
- **STANDARD_HEADER=[YES/NO]:** Defaults to `YES`. Whether to write standard header fields (i.e mod, src, chs, ver, ifv, dve, fft). If set to NO, only explicitly specified HEADER_xxx fields will be written.
- **HEADER_SRC=value:** Defaults to `UNKNOWN`. Value of the src header field.
- **HEADER_SRC_DATE=<DD.MM.YYYY>:** Defaults to `current` date (in GMT). Value of the date of the src header field as DD.MM.YYYY.
- **HEADER_SRC_TIME=<HH.MM.SS>:** Defaults to `current` time (in GMT). Value of the time of the src header field as HH.MM.SS.

- **HEADER_CHS=value:** Defaults to `ISO8859-1`. Value of the chs header field.
- **HEADER_VER=value:** Defaults to `1.4`. Value of the ver header field.
- **HEADER_IFV=value:** Defaults to `1.4`. Value of the ifv header field.
- **HEADER_DVE=value:** Defaults to `1.4`. Value of the dve header field.
- **HEADER_FFT=value:** Defaults to `''` (empty string). Value of the fft header field.
- **HEADER_xxx**=value:** Value of the *xxx* (user defined) header field.

7.80.3 Links

- VDV-451 file format (German)
- VDV-452 data model (German)
- Austrian INTREST data format (German)

7.81 VFK - Czech Cadastral Exchange Data Format

Driver short name

VFK

Build dependencies

libsqlite3

This driver reads VFK files, i.e. data in the *Czech cadastral exchange data format*. The VFK file is recognized as an datasource with multiple layers.

The driver is compiled only if GDAL is *built with SQLite support*.

Points are represented as wkbPoints, lines and boundaries as wkbLineStrings and areas as wkbPolygons. wkbMulti* features are not used. Feature types cannot be mixed in one layer.

7.81.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.81.2 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *`GDALOpenEx()`* (C) or *`gdal.OpenEx`* (Python). The following open options can be specified:

- **SUPPRESS_GEOMETRY=[YES/NO]:** Defaults to `NO`. Setting it to YES will skip resolving geometry. All layers will be recognized with no geometry type. Mostly useful when user is interested at attributes only. Note that suppressing geometry can cause significant performance gain when reading input VFK data by the driver.
- **FILE_FIELD=[YES/NO]:** Defaults to `NO`. Setting it to YES will append new field *VFK_FILENAME* containing name of source VFK file to all layers.

7.81.3 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The following configuration options are available:

- **OGR_VFK_DB_NAME=value:** Determine the name of the SQLite backend database used when reading VFK data. By default, SQLite database is created in a directory of input VFK file (with file extension '.db').
- **OGR_VFK_DB_OVERWRITE=[YES/NO]:** Determines whether the driver should overwrite an existing SQLite database and stores data read from input VFK file into newly created DB.
- **OGR_VFK_DB_DELETE=[YES/NO]:** Determines whether the driver should delete the backend SQLite database when closing the datasource.
- **OGR_VFK_DB_SPATIAL=[YES/NO]:** Determines whether the driver should store resolved geometries in the backend SQLite database. If `YES`, geometries are resolved only once when building SQLite database from VFK data. Geometries are stored in WKB format. Note that GDAL doesn't need to be built with SpatiaLite support. If `NO`, geometries are not stored in the DB and are resolved when reading data from DB on the fly.
- **OGR_VFK_DB_READ_ALL_BLOCKS=[YES/NO]:** Determines whether all data blocks should be read, or only the data blocks requested by the used.
- **OGR_VFK_DB_READ=[YES/NO]:** If `YES`, opening a VFK backend SQLite database will cause the VFK driver to be used instead of the SQLite driver.

7.81.3.1 Internal working and performance tweaking

If backend SQLite database already exists then the driver reads features directly from the database and not from input VFK file given as an input datasource. This causes significant performance gain when reading features by the driver.

The driver reads by default all data blocks from VFK file when building backend SQLite database. When configuration option *`OGR_VFK_DB_READ_ALL_BLOCKS`* =NO is given, the driver reads only data blocks which are requested by the user. This can be useful when the user want to process only part of VFK data.

7.81.3.2 Examples

Data related to a single cadastral area is typically distributed in multiple VFK files. Example below is based on sample VFK files provided by the Czech State Administration of Land Surveying and Cadastre. In order to process all VFK files related to a single cadastral area (in example below with ID 602515), the configuration option *`OGR_VFK_DB_NAME`* has to be defined.

```
# load first file mapa/602515.vfk
ogrinfo --config OGR_VFK_DB_NAME 602515.db mapa/602515.vfk
# load second file spi_s_jpv/602515.vfk
ogrinfo --config OGR_VFK_DB_NAME 602515.db spi_s_jpv/602515.vfk
# now we can access eg. geometry of parcels
ogrinfo 602515.db PAR -fid 1
...
```

7.81.4 Datasource name

Datasource name is a full path to the VFK file.

The driver supports reading files managed by VSI Virtual File System API, which include "regular" files, as well as files in the /vsizip/, /vsigzip/, and /vsicurl/ read-only domains.

A full path to the backend SQLite database can also be used as an datasource. By default, such datasource is read by SQLite driver. If configuration option *OGR_VFK_DB_READ* =YES is given, such datasource is opened by VFK driver instead.

7.81.5 Layer names

VFK data blocks are used as layer names.

7.81.6 Filters

7.81.6.1 Attribute filter

An internal SQL engine is used to evaluate the expression. Evaluation is done once when the attribute filter is set.

7.81.6.2 Spatial filter

Bounding boxes of features stored in topology structure are used to evaluate if a features matches current spatial filter. Evaluation is done once when the spatial filter is set.

7.81.7 References

- OGR VFK Driver Implementation Issues
- Czech cadastral exchange data format documentation (in Czech)

7.82 VRT -- OGR Virtual Format

Driver short name

VRT

Driver built-in by default

This driver is built-in by default

OGR Virtual Format is a driver that transforms features read from other drivers based on criteria specified in an XML control file. It is primarily used to derive spatial layers from flat tables with spatial information in attribute columns. It can also be used to associate coordinate system information with a datasource, merge layers from different datasources into a single data source, or even just to provide an anchor file for access to non-file oriented datasources.

Vector virtual files may be prepared by hand, however e.g., **ogrmerge**, such as in `ogrmerge -o merged.vrt *.shp` can also be used.

Note .vrt files starting with

- `<OGRVRTDataSource>` are managed by this driver.
- `<VRTDataset>` are managed by the *raster VRT* driver

7.82.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.82.2 Creation Issues

The CreateFeature(), SetFeature() and DeleteFeature() operations are supported on a layer of a VRT dataset, if the following conditions are met :

- the VRT dataset is opened in update mode
- the underlying source layer supports those operations
- the *SrcLayer* element is used (as opposed to the *SrcSQL* element)
- the FID of the VRT features is the same as the FID of the source features, that is to say, the *FID* element is not specified

7.82.3 Virtual File Format

The root element of the XML control file is **OGRVRTDataSource**. It has an **OGRVRTLayer** (or **OGRVRTWarpedLayer** or **OGRVRTUnionLayer**) child for each layer in the virtual datasource, and a **Metadata** element.

An XML schema of the OGR VRT format is provided in ogrvrt.xsd. When GDAL is configured with libXML2 support, that schema will be used to validate the VRT documents. Non-conformities will be reported only as warnings. That validation can be disabled by setting the `GDAL_XML_VALIDATION` configuration option to "NO".

7.82.3.1 Metadata element

Metadata (optional): This element contains a list of metadata name/value pairs associated with the dataset as a whole. It has <MDI> (metadata item) subelements which have a "key" attribute and the value as the data of the element. The Metadata element can be repeated multiple times, in which case it must be accompanied with a "domain" attribute to indicate the name of the metadata domain.

7.82.3.2 OGRVRTLayer element

A **OGRVRTLayer** element should have a **name** attribute with the layer name, and may have the following subelements:

- **SrcDataSource** (mandatory): The value is the name of the datasource that this layer will be derived from. The element may optionally have a **relativeToVRT** attribute which defaults to "0", but if "1" indicates that the source datasource should be interpreted as relative to the virtual file. This can be any OGR supported dataset, including ODBC, CSV, etc. The element may also have a **shared** attribute to control whether the datasource should be opened in shared mode. Defaults to OFF for SrcLayer use and ON for SrcSQL use.
- **OpenOptions** (optional): This element may list a number of open options as child elements of the form <OOI key="key_name">value_name</OOI>
- **Metadata** (optional): This element contains a list of metadata name/value pairs associated with the layer as a whole. It has <MDI> (metadata item) subelements which have a "key" attribute and the value as the data of the element. The Metadata element can be repeated multiple times, in which case it must be accompanied with a "domain" attribute to indicate the name of the metadata domain.

- **SrcLayer** (optional): The value is the name of the layer on the source data source from which this virtual layer should be derived. If this element isn't provided, then the SrcSQL element must be provided.
- **SrcSQL** (optional): An SQL statement to execute to generate the desired layer result. This should be provided instead of the SrcLayer for statement derived results. Some limitations may apply for SQL derived layers. An optional **dialect** attribute can be specified on the SrcSQL element to specify which SQL "dialect" should be used : possible values are currently *OGRSQL* or *SQLITE*. If *dialect* is not specified, the default dialect of the datasource will be used.
- **FID** (optional): Name of the source attribute column from which the FID of features should be derived. If not provided, the FID of the source features will be used directly.

  Different situations are possible:

  - `<FID>source_field_name</FID>`

    A FID column will be reported as source_field_name with the content of source field source_field_name.

  - `<FID name="dest_field_name">source_field_name</FID>`

    A FID column will be reported as dest_field_name with the content of source field source_field_name. dest_field_name can potentially be set to the empty string.

  - `<FID />`

    No FID column is reported. The FID value of VRT features is the FID value of the source features.

  - `<FID name="dest_field_name"/>`

    A FID column will be reported as dest_field_name with the content of the implicit source FID column. The FID value of VRT features is the FID value of the source features.
- **Style** (optional): Name of the attribute column from which the style of features should be derived. If not provided, the style of the source features will be used directly.
- **GeometryType** (optional): The geometry type to be assigned to the layer. If not provided it will be taken from the source layer. The value should be one of "wkbNone", "wkbUnknown", "wkbPoint", "wkbLineString", "wkbPolygon", "wkbMultiPoint", "wkbMultiLineString", "wkbMultiPolygon", or "wkbGeometryCollection". Optionally "25D" may be appended to mark it as including Z coordinates. Defaults to "wkbUnknown" indicating that any geometry type is allowed.
- **LayerSRS** (optional): The value of this element is the spatial reference to use for the layer. If not provided, it is inherited from the source layer. The value may be WKT or any other input that is accepted by the OGRSpatialReference::SetUserInput() method. If the value is NULL, then no SRS will be used for the layer.
- **GeometryField** (optional): This element is used to define how the geometry for features should be derived.

  The GeometryField element can be repeated as many times as necessary to create multiple geometry fields. If no **GeometryField** element is specified, all the geometry fields of the source layer will be exposed by the VRT layer. In order not to expose any geometry field of the source layer, you need to specify OGRVRTLayer-level **GeometryType** element to wkbNone.

  The following attributes can be defined:

  - **name** = string (recommended, and mandatory if the VRT will expose multiple geometry fields)

    Name that will be used to define the VRT geometry field name. If not set, empty string is used.

  - **encoding** = Direct/WKT/WKB/PointFromColumns (optional)

    Type of geometry encoding.

If the encoding is "Direct" or not specified, then the **field** attribute must be set to the name of the source geometry field, if there are multiple source geometry fields. If neither **encoding** nor **field** are specified, it is assumed that the name of source geometry field is the value of the **name** attribute.

If the encoding is "WKT" or "WKB" then the **field** attribute must be set to the name of the source field containing the WKT or WKB geometry.

If the encoding is "PointFromColumns" then the **x**, **y**, **z** and **m** attributes must be set to the names of the columns to be used for the X, Y, Z and M coordinates. The **z** and **m** attributes are optional.

– **field** = string (conditional)

Name of the source field (or source geometry field for **encoding** = Direct) from which this GeometryField should fetch geometries. This must be set if **encoding** is WKT or WKB.

– **x**, **y**, **z**, **m** = string (conditional)

Name of the source fields for the X, Y, Z and M coordinates when **encoding** = PointFromColumns

– **reportSrcColumn** = true/false (optional)

Specify whether the source geometry fields (the fields set in the **field**, **x**, **y**, **z**, **m** attributes) should also be included as fields of the VRT layer. It defaults to true. If set to false, the source geometry fields will only be used to build the geometry of the features of the VRT layer.

Note that reportSrcColumn=true is taken into account only if no explicit **Field** element is defined and when **encoding** is not "Direct". If at least one field is explicitly defined, and reporting of the source geometry field is desired, an explicit **Field** element for it must be defined.

– **nullable** = true/false (optional)

The optional **nullable** attribute can be used to specify whether the geometry field is nullable. It defaults to "true".

The following child elements of **GeometryField** can be defined:

– **GeometryType** (optional) : same syntax as OGRVRTLayer-level **GeometryType**. Useful when there are multiple geometry fields.

– **SRS** (optional) : same syntax as OGRVRTLayer-level **LayerSRS** (note SRS vs LayerSRS). Useful when there are multiple geometry fields.

– **SrcRegion** (optional) : same syntax as OGRVRTLayer-level **SrcRegion**. Useful when there are multiple geometry fields.

– **ExtentXMin**, **ExtentYMin**, **ExtentXMax** and **ExtentXMax** (optional) : same syntax as OGRVRTLayer-level elements of same name. Useful when there are multiple geometry fields.

– **XYResolution** (optional, GDAL >= 3.9): Resolution for the coordinate precision of the X and Y coordinates. Expressed in the units of the X and Y axis of the SRS

– **ZResolution** (optional, GDAL >= 3.9): Resolution for the coordinate precision of the Z coordinates. Expressed in the units of the Z axis of the SRS

– **MResolution** (optional, GDAL >= 3.9): Resolution for the coordinate precision of the M coordinates.

- **SrcRegion** (optional) : This element is used to define an initial spatial filter for the source features, such that only features intersecting `SrcRegion` will be returned in the VRT layer. This spatial filter will be combined with any spatial filter explicitly set on the VRT layer with the SetSpatialFilter() method. The value of the element must be a valid WKT string defining a geometry in the spatial reference system of the source layer. An optional **clip** attribute can be set to "TRUE" to clip the geometries to the source region, otherwise the source geometries are not modified.

Field (optional): One or more attribute fields may be defined with Field elements. If no Field elements are defined, the fields of the source layer/sql will be defined on the VRT layer. The Field may have the following attributes:

- **name** (required): the name of the field.
- **type**: the field type, one of "Integer", "IntegerList", "Real", "RealList", "String", "StringList", "Binary", "Date", "Time", or "DateTime". Defaults to "String".
- **subtype**: the field subtype, one of "None", "Boolean", "Int16", "Float32". Defaults to "None".
- **width**: the field width. Defaults to unknown.
- **precision**: the field width. Defaults to zero.
- **src**: the name of the source field to be copied to this one. Defaults to the value of "name".
- **nullable** can be used to specify whether the field is nullable. It defaults to "true".
- **unique** can be used to specify whether the field has a unique constraint. It defaults to "false". (GDAL >= 3.2)
- **alternativeName**: the field alternative name. (GDAL >= 3.7)
- **comment**: the field comment. (GDAL >= 3.7)

- **FeatureCount** (optional) : This element is used to define the feature count of the layer (when no spatial or attribute filter is set). This can be useful on static data, when getting the feature count from the source layer is slow.
- **ExtentXMin**, **ExtentYMin**, **ExtentXMax** and **ExtentXMax** (optional) : Those elements are used to define the extent of the layer. This can be useful on static data, when getting the extent from the source layer is slow.

7.82.3.3 OGRVRTWarpedLayer element

A **OGRVRTWarpedLayer** element is used to do on-the-fly reprojection of a source layer. It may have the following subelements:

- **OGRVRTLayer**, **OGRVRTWarpedLayer** or **OGRVRTUnionLayer** (mandatory): the source layer to reproject.
- **SrcSRS** (optional): The value of this element is the spatial reference to use for the layer before reprojection. If not specified, it is deduced from the source layer.
- **TargetSRS** (mandatory): The value of this element is the spatial reference to use for the layer after reprojection.
- **ExtentXMin**, **ExtentYMin**, **ExtentXMax** and **ExtentXMax** (optional) : Those elements are used to define the extent of the layer. This can be useful on static data, when getting the extent from the source layer is slow.
- **WarpedGeomFieldName** (optional) : The value of this element is the geometry field name of the source layer to wrap. If not specified, the first geometry field will be used. If there are several geometry fields, only the one matching WarpedGeomFieldName will be warped; the other ones will be untouched.

7.82.3.4 OGRVRTUnionLayer element

A **OGRVRTUnionLayer** element is used to concatenate the content of source layers. It should have a **name** and may have the following subelements:

- **OGRVRTLayer**, **OGRVRTWarpedLayer** or **OGRVRTUnionLayer** (mandatory and may be repeated): a source layer to add in the union.
- **PreserveSrcFID** (optional) : may be ON or OFF. If set to ON, the FID from the source layer will be used, otherwise a counter will be used. Defaults to OFF.

- **SourceLayerFieldName** (optional) : if specified, an additional field (named with the value of SourceLayerFieldName) will be added in the layer field definition. For each feature, the value of this field will be set with the name of the layer from which the feature comes from.
- **GeometryType** (optional) : see above for the syntax. If not specified, the geometry type will be deduced from the geometry type of all source layers.
- **LayerSRS** (optional) : see above for the syntax. If not specified, the SRS will be the SRS of the first source layer.
- **FieldStrategy** (optional, exclusive with **Field** or **GeometryField**) : may be **FirstLayer** to use the fields from the first layer found, **Union** to use a super-set of all the fields from all source layers, or **Intersection** to use a sub-set of all the common fields from all source layers. Defaults to **Union**.
- **Field** (optional, exclusive with **FieldStrategy**) : see above for the syntax. Note: the src attribute is not supported in the context of a OGRVRTUnionLayer element (field names are assumed to be identical).
- **GeometryField** (optional, exclusive with **FieldStrategy**): the **name** attribute and the following sub-elements **GeometryType**, **SRS** and **Extent[X|Y][Min|Max]** are available.
- **FeatureCount** (optional) : see above for the syntax
- **ExtentXMin**, **ExtentYMin**, **ExtentXMax** and **ExtentXMax** (optional) : see above for the syntax

7.82.4 Examples

7.82.4.1 Example 1: ODBC Point Layer

In the following example (disease.ovf) the worms table from the ODBC database DISEASE is used to form a spatial layer. The virtual file uses the "x" and "y" columns to get the spatial location. It also marks the layer as a point layer, and as being in the WGS84 coordinate system.

```
<OGRVRTDataSource>
    <OGRVRTLayer name="worms">
        <SrcDataSource>ODBC:DISEASE,worms</SrcDataSource>
        <SrcLayer>worms</SrcLayer>
        <GeometryType>wkbPoint</GeometryType>
        <LayerSRS>WGS84</LayerSRS>
        <GeometryField encoding="PointFromColumns" x="x" y="y"/>
    </OGRVRTLayer>
</OGRVRTDataSource>
```

7.82.4.2 Example 2: Renaming attributes

It can be useful in some circumstances to be able to rename the field names from a source layer to other names. This is particularly true when you want to transcode to a format whose schema is fixed, such as GPX (<name>, <desc>, etc.). This can be accomplished using SQL this way:

```
<OGRVRTDataSource>
    <OGRVRTLayer name="remapped_layer">
        <SrcDataSource>your_source.shp</SrcDataSource>
        <SrcSQL>SELECT src_field_1 AS name, src_field_2 AS desc FROM your_source_layer_
↪name</SrcSQL>
    </OGRVRTLayer>
</OGRVRTDataSource>
```

This can also be accomplished using explicit field definitions:

```
<OGRVRTDataSource>
    <OGRVRTLayer name="remapped_layer">
        <SrcDataSource>your_source.shp</SrcDataSource>
        <SrcLayer>your_source</SrcLayer>
        <Field name="name" src="src_field_1" />
        <Field name="desc" src="src_field_2" type="String" width="45" />
    </OGRVRTLayer>
</OGRVRTDataSource>
```

7.82.4.3 Example 3: Transparent spatial filtering

The following example will only return features from the source layer that intersect the (0,40)-(10,50) region. Furthermore, returned geometries will be clipped to fit into that region.

```
<OGRVRTDataSource>
    <OGRVRTLayer name="source">
        <SrcDataSource>source.shp</SrcDataSource>
        <SrcRegion clip="true">POLYGON((0 40,10 40,10 50,0 50,0 40))</SrcRegion>
    </OGRVRTLayer>
</OGRVRTDataSource>
```

7.82.4.4 Example 4: Reprojected layer

The following example will return the source.shp layer reprojected to EPSG:4326.

```
<OGRVRTDataSource>
    <OGRVRTWarpedLayer>
        <OGRVRTLayer name="source">
            <SrcDataSource>source.shp</SrcDataSource>
        </OGRVRTLayer>
        <TargetSRS>EPSG:4326</TargetSRS>
    </OGRVRTWarpedLayer>
</OGRVRTDataSource>
```

7.82.4.5 Example 5: Union layer

The following example will return a layer that is the concatenation of source1.shp and source2.shp.

```
<OGRVRTDataSource>
    <OGRVRTUnionLayer name="unionLayer">
        <OGRVRTLayer name="source1">
            <SrcDataSource>source1.shp</SrcDataSource>
        </OGRVRTLayer>
        <OGRVRTLayer name="source2">
            <SrcDataSource>source2.shp</SrcDataSource>
        </OGRVRTLayer>
    </OGRVRTUnionLayer>
</OGRVRTDataSource>
```

7.82.4.6 Example 6: SQLite/Spatialite SQL dialect

The following example will return four different layers which are generated in a fly from the same polygon shapefile. The first one is the shapefile layer as it stands. The second layer gives simplified polygons by applying SpatiaLite function "Simplify" with parameter tolerance=10. In the third layer the original geometries are replaced by their convex hulls. In the fourth layer SpatiaLite function PointOnSurface is used for replacing the original geometries by points which are inside the corresponding source polygons. Note that for using the last three layers of this VRT file GDAL must be compiled with SQLite and SpatiaLite.

```
<OGRVRTDataSource>
    <OGRVRTLayer name="polygons">
        <SrcDataSource>polygons.shp</SrcDataSource>
    </OGRVRTLayer>
    <OGRVRTLayer name="polygons_as_simplified">
        <SrcDataSource>polygons.shp</SrcDataSource>
        <SrcSQL dialect="sqlite">SELECT Simplify(geometry,10) from polygons</SrcSQL>
    </OGRVRTLayer>
    <OGRVRTLayer name="polygons_as_hulls">
        <SrcDataSource>polygons.shp</SrcDataSource>
        <SrcSQL dialect="sqlite">SELECT ConvexHull(geometry) from polygons</SrcSQL>
    </OGRVRTLayer>
    <OGRVRTLayer name="polygons_as_points">
        <SrcDataSource>polygons.shp</SrcDataSource>
        <SrcSQL dialect="sqlite">SELECT PointOnSurface(geometry) from polygons</SrcSQL>
    </OGRVRTLayer>
</OGRVRTDataSource>
```

7.82.4.7 Example 7: Multiple geometry fields

The following example will expose all the attribute and geometry fields of the source layer:

```
<OGRVRTDataSource>
    <OGRVRTLayer name="test">
        <SrcDataSource>PG:dbname=testdb</SrcDataSource>
    </OGRVRTLayer>
</OGRVRTDataSource>
```

To expose only part (or all!) of the fields:

```
<OGRVRTDataSource>
    <OGRVRTLayer name="other_test">
        <SrcDataSource>PG:dbname=testdb</SrcDataSource>
        <SrcLayer>test</SrcLayer>
        <GeometryField name="pg_geom_field_1" />
        <GeometryField name="vrt_geom_field_2" field="pg_geom_field_2">
            <GeometryType>wkbPolygon</GeometryType>
            <SRS>EPSG:4326</SRS>
            <ExtentXMin>-180</ExtentXMin>
            <ExtentYMin>-90</ExtentYMin>
            <ExtentXMax>180</ExtentXMax>
            <ExtentYMax>90</ExtentYMax>
        </GeometryField>
        <Field name="vrt_field_1" src="src_field_1" />
    </OGRVRTLayer>
```

(continues on next page)

(continued from previous page)

```
</OGRVRTDataSource>
```

To reproject the 'pg_geom_field_2' geometry field to EPSG:4326:

```
<OGRVRTDataSource>
    <OGRVRTWarpedLayer>
        <OGRVRTLayer name="other_test">
            <SrcDataSource>PG:dbname=testdb</SrcDataSource>
        </OGRVRTLayer>
        <WarpedGeomFieldName>pg_geom_field_2</WarpedGeomFieldName>
        <TargetSRS>EPSG:32631</TargetSRS>
    </OGRVRTWarpedLayer>
</OGRVRTDataSource>
```

To make the union of several multi-geometry layers and keep only a few of them:

```
<OGRVRTDataSource>
    <OGRVRTUnionLayer name="unionLayer">
        <OGRVRTLayer name="source1">
            <SrcDataSource>PG:dbname=testdb</SrcDataSource>
        </OGRVRTLayer>
        <OGRVRTLayer name="source2">
            <SrcDataSource>PG:dbname=testdb</SrcDataSource>
        </OGRVRTLayer>
        <GeometryField name="pg_geom_field_2">
            <GeometryType>wkbPolygon</GeometryType>
            <SRS>EPSG:4326</SRS>
            <ExtentXMin>-180</ExtentXMin>
            <ExtentYMin>-90</ExtentYMin>
            <ExtentXMax>180</ExtentXMax>
            <ExtentYMax>90</ExtentYMax>
        </GeometryField>
        <GeometryField name="pg_geom_field_3" />
        <Field name="src_field_1" />
    </OGRVRTUnionLayer>
</OGRVRTDataSource>
```

7.82.5 Other Notes

- When the *GeometryField* is "WKT" spatial filtering is applied after extracting all rows from the source datasource. Essentially that means there is no fast spatial filtering on WKT derived geometries.
- When the *GeometryField* is "PointFromColumns", and a *SrcLayer* (as opposed to *SrcSQL*) is used, and a spatial filter is in effect on the virtual layer then the spatial filter will be internally translated into an attribute filter on the X and Y columns in the *SrcLayer*. In cases where fast spatial filtering is important it can be helpful to index the X and Y columns in the source datastore, if that is possible. For instance if the source is an RDBMS. You can turn off that feature by setting the *useSpatialSubquery* attribute of the GeometryField element to FALSE.
- .vrt files starting with - <OGRVRTDataSource> open with ogrinfo, etc. - <VRTDataset> open with gdalinfo, etc.

7.83 WAsP - WAsP .map format

Driver short name

WAsP

Driver built-in by default

This driver is built-in by default

This driver writes .map files to be used with WAsP. The only allowed geometries are linestrings.

7.83.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.83.2 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). The following configuration options are available:

- **WASP_FIELDS=value:** a comma separated list of fields. For elevation, the name of the height field. For roughness, the name of the left and right roughness fields resp.
- **WASP_MERGE=value:** this may be set to "NO". Used only when generating roughness from polygons. All polygon boundaries will be output (including those with the same left and right roughness). This is useful (along with option -skipfailures) for debugging incorrect input geometries.
- **WASP_GEOM_FIELD=value:** in case input has several geometry columns and the first one (default) is not the right one.
- **WASP_TOLERANCE=value:** specify a tolerance for line simplification of output (calls geos).
- **WASP_ADJ_TOLER=value:** points that are less than tolerance apart from previous point on x and on y are omitted.
- **WASP_POINT_TO_CIRCLE_RADIUS=value:** lines that became points due to simplification are replaces by 8 point circles (octagons).

Note that if not option is specified, the layer is assumed to be an elevation layer where the elevation is the z-components of the linestrings' points.

7.84 WFS - OGC WFS service

Driver short name

WFS

Build dependencies

libcurl

This driver can connect to a OGC WFS service. It supports WFS 1.0, 1.1 and 2.0 protocols. GDAL/OGR must be built with Curl support in order for the WFS driver to be compiled. Usually WFS requests return results in GML format, so the GML driver should generally be set-up for read support (thus requiring GDAL/OGR to be built with Xerces or Expat support). It is sometimes possible to use alternate underlying formats when the server supports them (such as OUTPUTFORMAT=json).

The driver supports read-only services, as well as transactional ones (WFS-T).

7.84.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

7.84.2 Dataset name syntax

The minimal syntax to open a WFS datasource is : *WFS:http://path/to/WFS/service* or *http://path/to/WFS/service?SERVICE=WFS*

Starting with GDAL 3.10, specifying the `-if WFS` option to command line utilities accepting it, or `WFS` as the only value of the `papszAllowedDrivers` of *`GDALOpenEx()`*, also forces the driver to recognize the passed URL without the `WFS:` prefix.

Additional optional parameters can be specified such as *TYPENAME*, *VERSION*, *MAXFEATURES* (WFS < 2) or *COUNT* (WFS > 2) as specified in WFS specification.

The name provided to the TYPENAME parameter must be exactly the layer name reported by OGR, in particular with its namespace prefix when its exists. Note: several type names can be provided and separated by comma.

It is also possible to specify the name of an XML file whose content matches the following syntax (the <OGRWFS-DataSource> element must be the first bytes of the file):

```
<OGRWFSDataSource>
    <URL>http://path/to/WFS/service[?OPTIONAL_PARAMETER1=VALUE[&amp;OPTIONAL_
↪PARAMETER2=VALUE]]</URL>
</OGRWFSDataSource>
```

Note: the URL must be XML-escaped, for example the & character must be written as *&*

At the first opening, the content of the result of the *GetCapabilities* request will be appended to the file, so that it can be cached for later openings of the dataset. The same applies for the *DescribeFeatureType* request issued to discover the field definition of each layer.

The service description file has the following additional elements as immediate children of the `OGRWFSDataSource` element that may be optionally set.

- **Timeout**: The timeout to use for remote service requests. If not provided, the libcurl default is used.
- **UserPwd**: May be supplied with *userid:password* to pass a userid and password to the remote server.
- **HttpAuth**: May be BASIC, NTLM or ANY to control the authentication scheme to be used.
- **Version**: Set a specific WFS version to use (either 1.0.0 or 1.1.0).
- **PagingAllowed**: Set to ON if paging must be enabled. See "Request paging" section.
- **PageSize**: Page size when paging is enabled. See "Request paging" section.
- **BaseStartIndex**: Base of the start index when paging is enabled (0 or 1). See "Request paging" section.
- **COOKIE**: HTTP cookies that are passed in HTTP requests, formatted as COOKIE1=VALUE1; COOKIE2=VALUE2... Additional HTTP headers can be sent by setting the *GDAL_HTTP_HEADER_FILE* configuration option to point to a filename of a text file with "key: value" HTTP headers.

7.84.3 Request paging

The WFS driver will read the GML content as a stream instead as a whole file, which will improve interactivity and help when the content cannot fit into memory. This can be turned off by setting the *OGR_WFS_USE_STREAMING* configuration option to NO if this is not desirable (for example, when iterating several times on a layer that can fit into memory). When streaming is enabled, GZip compression is also requested. It has been observed that some WFS servers, that cannot do on-the-fly compression, will cache on their side the whole content to be sent before sending the first bytes on the wire. To avoid this, you can set the *CPL_CURL_GZIP* configuration option to NO.

7.84.3.1 Paging with WFS 2.0

The WFS driver will automatically detect if server supports paging, when requesting a WFS 2.0 server. The page size (number of features fetched in a single request) is limited to 100 by default when not declared by the server. It can be changed by setting the *OGR_WFS_PAGE_SIZE* configuration option, or by specifying COUNT as a query parameter in the URL of the connection string.

If only the N first features must be downloaded and paging through the whole layer is not desirable, the *OGR_WFS_PAGING_ALLOWED* configuration option should be set to OFF.

Starting with GDAL 3.8, if the feature count is already known before fetching the first feature, and the feature count is less than the page size, then the WFS driver will not use paging (no `STARTINDEX` parameter will be used in the requests). (This behavior improves compatibility with Geoserver when datasources with no primary key.)

7.84.3.2 Paging with WFS 1.0 or 1.1

Some servers (such as MapServer >= 6.0) support the use of STARTINDEX that allows doing the requests per "page", and thus to avoid downloading the whole content of the layer in a single request. Paging was introduced in WFS 2.0.0 but servers may support it as an vendor specific option also with WFS 1.0.0 and 1.1.0. The OGR WFS client will use paging when the *OGR_WFS_PAGING_ALLOWED* configuration option is explicitly set to ON. The page size (number of features fetched in a single request) is limited to 100 by default when not declared by the server. It can be changed by setting the *OGR_WFS_PAGE_SIZE* configuration option.

WFS 2.0.2 specification has clarified that the first feature in paging is at index 0. But some server implementations of WFS paging have considered that it was at index 1 (including MapServer <= 6.2). The default base start index is 0, as mandated by the specification. The *OGR_WFS_BASE_START_INDEX* configuration option can however be set to 1 to be compatible with the server implementations that considered the first feature to be at index 1.

7.84.3.3 Paging options

Those 3 options (*OGR_WFS_PAGING_ALLOWED*, *OGR_WFS_PAGE_SIZE*, *OGR_WFS_BASE_START_INDEX*) can also be set in a WFS XML description file with the elements of similar names (PagingAllowed, PageSize, BaseStartIndex).

7.84.4 Filtering

The driver will forward any spatial filter set with SetSpatialFilter() to the server. It also makes its best effort to do the same for attribute filters set with SetAttributeFilter() when possible (turning OGR SQL language into OGC filter description). When this is not possible, it will default to client-side only filtering, which can be a slow operation because involving fetching all the features from the servers.

The following spatial functions can be used :

- the 8 spatial binary predicate: **ST_Equals, ST_Disjoint, ST_Touches, ST_Contains, ST_Intersects, ST_Within, ST_Crosses and ST_Overlaps** that take 2 geometry arguments. Typically the geometry column name, and a constant geometry such as built with ST_MakeEnvelope or ST_GeomFromText.
- **ST_DWithin(geom1,geom2,distance_in_meters)**
- **ST_Beyond(geom1,geom2,distance_in_meters)**
- **ST_MakeEnvelope(xmin,ymin,xmax,ymax[,srs])**: to build an envelope. srs can be an integer (an EPSG code), or a string directly set as the srsName attribute of the gml:Envelope. GDAL will take care of needed axis swapping, so coordinates should be expressed in the "natural GIS order" (for example long,lat for geodetic systems)
- **ST_GeomFromText(wkt,[srs])**: to build a geometry from its WKT representation.

Note that those spatial functions are only supported as server-side filters.

7.84.5 Layer joins

For WFS 2.0 servers that support joins, SELECT statements that involve joins will be run on server side. Spatial joins can also be done by using the above mentioned spatial functions, if the server supports spatial joins.

There might be restrictions set by server on the complexity of the joins. The OGR WFS driver also restricts column selection to be column names, potentially with aliases and type casts, but not expressions. The ON and WHERE clauses must also be evaluated on server side, so no OGR special fields are allowed for example. ORDER BY clauses are supported, but the fields must belong to the primary table.

Example of valid statement :

```
SELECT t1.id, t1.val1, t1.geom, t2.val1 FROM my_table AS t1 JOIN another_table AS t2 ON␣
↪t1.id = t2.t1id
```

or

```
SELECT * FROM my_table AS t1 JOIN another_table AS t2 ON ST_Intersects(t1.geom, t2.geom)
```

7.84.6 Write support / WFS-T

The WFS-T protocol only enables the user to operate at feature level. No datasource, layer or field creations are possible.

Write support is only enabled when the datasource is opened in update mode.

The mapping between the operations of the WFS Transaction service and the OGR concepts is the following:

- OGRFeature::CreateFeature() <==> WFS insert operation
- OGRFeature::SetFeature() <==> WFS update operation
- OGRFeature::DeleteFeature() <==> WFS delete operation

Lock operations (LockFeature service) are not available at that time.

There are a few caveats to keep in mind. OGR feature ID (FID) is an integer based value, whereas WFS/GML gml:id attribute is a string. Thus it is not always possible to match both values. The WFS driver exposes then the gml:id attribute of a feature as a 'gml_id' field.

When inserting a new feature with CreateFeature(), and if the command is successful, OGR will fetch the returned gml:id and set the 'gml_id' field of the feature accordingly. It will also try to set the OGR FID if the gml:id is of the form layer_name.numeric_value. Otherwise the FID will be left to its unset default value.

When updating an existing feature with SetFeature(), the OGR FID field will be ignored. The request issued to the driver will only take into account the value of the gml:id field of the feature. The same applies for DeleteFeature().

7.84.7 Write support and OGR transactions

The above operations are by default issued to the server synchronously with the OGR API call. This however can cause performance penalties when issuing a lot of commands due to many client/server exchanges.

It is possible to surround those operations between OGRLayer::StartTransaction() and OGRLayer::CommitTransaction(). The operations will be stored into memory and only executed at the time CommitTransaction() is called.

The drawback for CreateFeature() is that the user cannot know which gml:id have been assigned to the inserted features. A special SQL statement has been introduced into the WFS driver to workaround this : by issuing the "SELECT _LAST_INSERTED_FIDS_ FROM layer_name" (where layer_name is to be replaced with the actual layer_name) command through the OGRDataSource::ExecuteSQL(), a layer will be returned with as many rows with a single attribute gml_id as the count of inserted features during the last committed transaction.

Note : currently, only CreateFeature() makes use of OGR transaction mechanism. SetFeature() and DeleteFeature() will still be issued immediately.

7.84.8 Special SQL commands

The following SQL / pseudo-SQL commands passed to OGRDataSource::ExecuteSQL() are specific of the WFS driver :

- "DELETE FROM layer_name WHERE expression" : this will result into a WFS delete operation. This can be a fast way of deleting one or several features. In particularly, this can be a faster replacement for OGRLayer::DeleteFeature() when the gml:id is known, but the feature has not been fetched from the server.
- "SELECT _LAST_INSERTED_FIDS_ FROM layer_name" : see above paragraph.

Currently, any other SQL command will be processed by the generic layer, meaning client-side only processing. Server side spatial and attribute filtering must be done through the SetSpatialFilter() and SetAttributeFilter() interfaces.

7.84.9 Special layer : WFSLayerMetadata

A "hidden" layer called "WFSLayerMetadata" is filled with records with metadata for each WFS layer.

Each record contains a "layer_name", "title" and "abstract" field, from the document returned by GetCapabilities.

That layer is returned through GetLayerByName("WFSLayerMetadata").

7.84.10 Special layer : WFSGetCapabilities

A "hidden" layer called "WFSGetCapabilities" is filled with the raw XML result of the GetCapabilities request.

That layer is returned through GetLayerByName("WFSGetCapabilities").

7.84.11 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are available:

- **URL=value:** URL to the WFS server endpoint. Required when using the "WFS:" string as the connection string.
- **TRUST_CAPABILITIES_BOUNDS=[YES/NO]:** Defaults to `NO`. Whether to trust layer bounds declared in GetCapabilities response, for faster GetExtent() runtime.
- **EMPTY_AS_NULL=[YES/NO]:** Defaults to `YES`. By default (*EMPTY_AS_NULL=YES*), fields with empty content will be reported as being NULL, instead of being an empty string. This is the historic behavior. However this will prevent such fields to be declared as not-nullable if the application schema declared them as mandatory. So this option can be set to NO to have both empty strings being report as such, and mandatory fields being reported as not nullable.
- **INVERT_AXIS_ORDER_IF_LAT_LONG=[YES/NO]:** Defaults to `YES`. Whether to present SRS and coordinate ordering in traditional GIS order.
- **CONSIDER_EPSG_AS_URN=[YES/NO/AUTO]:** Defaults to `AUTO`. Whether to consider srsName like EPSG:XXXX as respecting EPSG axis order.
- **EXPOSE_GML_ID=[YES/NO]:** Defaults to `YES`. Whether to expose the gml:id attribute of a GML feature as the gml_id OGR field. Note that hiding gml_id will prevent WFS-T from working.

7.84.12 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *CPLSetConfigOption()* (C) or *gdal.config_options* (Python). The following configuration options are available:

- **OGR_WFS_USE_STREAMING=[YES/NO]:** Defaults to `YES`. Set to `NO` to disable streaming. See *Request paging*.
- **OGR_WFS_PAGE_SIZE=value:** Control the number of features fetched in a single request.
- **OGR_WFS_PAGING_ALLOWED=[ON/OFF]:** Set to `NO` to prevent paging through the whole layer.
- **OGR_WFS_BASE_START_INDEX=<integer>:** Sets the index of the first feature in paging.

7.84.13 Examples

7.84.13.1 Example 1: Listing the types of a WFS server

```
ogrinfo -ro WFS:https://www.wfs.nrw.de/geobasis/wfs_nw_dvg
```

7.84.13.2 Example 2: Listing the types of a WFS server whose layer structures are cached in a XML file

```
ogrinfo -ro wfs_nw_dvg.xml
```

7.84.13.3 Example 3: Listing the features of the dvg:nw_dvg2_rbz layer, with a spatial filter

```
ogrinfo -ro WFS:https://www.wfs.nrw.de/geobasis/wfs_nw_dvg dvg:nw_dvg2_rbz -spat 0 0␣
↪319874 5686804
```

7.84.13.4 Example 4: Retrieving the features of GN "Köln" and "Viersen" from the dvg:nw_dvg2_krs layer

```
ogrinfo WFS:https://www.wfs.nrw.de/geobasis/wfs_nw_dvg dvg:nw_dvg2_krs -ro -al -where
↪"GN='Köln' or GN='Viersen'"
```

7.84.13.5 Example 5: Displaying layer metadata

```
ogrinfo -ro -al WFS:https://www.wfs.nrw.de/geobasis/wfs_nw_dvg WFSLayerMetadata
```

7.84.14 See Also

- OGC WFS Standard
- *GML driver documentation*
- *OGC API - Features driver documentation*

7.85 XLS - MS Excel format

Driver short name

XLS

Build dependencies

libfreexl

This driver reads spreadsheets in the (old) Microsoft Excel binary format. This format pre-dates Excel Open XML Spreadsheet, introduced in Microsoft Excel 2007, and supported by the *XLSX* driver.

GDAL/OGR must be built against the FreeXL library (GPL/LPL/MPL licensed), and the driver has the same restrictions as the FreeXL library itself as far as which and how Excel files are supported, in particular formulas are not supported.

Each sheet is presented as a OGR layer. No geometry support is available directly (but you may use the OGR VRT capabilities for that).

7.85.1 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The following configuration options are available:

- **OGR_XLS_HEADERS=[FORCE/DISABLE/AUTO]:** Defaults to `AUTO`. By default, the driver will read the first lines of each sheet to detect if the first line might be the name of columns. If set to FORCE, the driver will consider the first line will be taken as the header line. If set to DISABLE, it will be considered as the first feature. Otherwise auto-detection will occur.
- **OGR_XLS_FIELD_TYPES=[STRING/AUTO]:** Defaults to `AUTO`. By default, the driver will try to detect the data type of fields. If set to STRING, all fields will be of String type.

7.85.2 See Also

- Homepage of the FreeXL library

7.86 XLSX - MS Office Open XML spreadsheet

Driver short name

XLSX

Build dependencies

libexpat

This driver can read, write and update spreadsheets in Microsoft Office Open XML (a.k.a. OOXML) spreadsheet format, generated by applications like Microsoft Office 2007 and later versions. LibreOffice/OpenOffice can also export documents in that format since their v3 version.

The driver is only available if GDAL/OGR is compiled against the Expat library.

Each sheet is presented as a OGR layer. No geometry support is available directly (but you may use the OGR VRT capabilities for that).

Note 1 : depending on the application that produced the file, the driver might succeed or not in retrieving the result of formulas. Some applications write the evaluated result of formulas in the document, in which case the driver will be able to retrieve it. Otherwise the raw formula string will be returned.

Note 2 : spreadsheets with passwords are not supported.

Note 3 : when updating an existing document, all existing styles, formatting, formulas and other concepts (charts, drawings, macros, ...) not understood by OGR will be lost : the document is re-written from scratch from the OGR data model.

7.86.1 Driver capabilities

Supports Create()

This driver supports the *GDALDriver::Create()* operation

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

7.86.2 Open options

Open options can be specified in command-line tools using the syntax `-oo <NAME>=<VALUE>` or by providing the appropriate arguments to *GDALOpenEx()* (C) or *gdal.OpenEx* (Python). The following open options are available:

- **HEADERS=[FORCE/DISABLE/AUTO]:** (GDAL >= 3.8) Defaults to `AUTO`. By default, the driver will read the first lines of each sheet to detect if the first line might be the name of columns. If set to FORCE, the driver will consider the first line as the header line. If set to DISABLE, it will be considered as the first feature. Otherwise auto-detection will occur.

- **FIELD_TYPES=[STRING/AUTO]:** (GDAL >= 3.8) Defaults to `AUTO`. By default, the driver will try to detect the data type of fields. If set to STRING, all fields will be of String type.

7.86.3 Configuration options

Configuration options can be specified in command-line tools using the syntax `--config <NAME>=<VALUE>` or using functions such as *`CPLSetConfigOption()`* (C) or *`gdal.config_options`* (Python). The following configuration options are available:

- **OGR_XLSX_HEADERS=[FORCE/DISABLE/AUTO]:** Defaults to `AUTO`. By default, the driver will read the first lines of each sheet to detect if the first line might be the name of columns. If set to FORCE, the driver will consider the first line as the header line. If set to DISABLE, it will be considered as the first feature. Otherwise auto-detection will occur.
- **OGR_XLSX_FIELD_TYPES=[STRING/AUTO]:** Defaults to `AUTO`. By default, the driver will try to detect the data type of fields. If set to STRING, all fields will be of String type.

7.86.4 Examples

7.86.4.1 Example 1

Transform CSV file input.csv to an Office Open XML file output.xlsx. The spreadsheet will contain one sheet with name "input".

```
ogr2ogr -f "XLSX" output.xlsx input.csv
```

Add the contents of CSV file input2.csv to the OOXML file output.xlsx created in the first example. The spreadsheet will now contain two sheets, "input" and "input2".

```
ogr2ogr -f "XLSX" output.xlsx -update input2.csv
```

7.87 XODR -- OpenDRIVE Road Description Format

Added in version 3.10.

Driver short name

XODR

Build dependencies

libOpenDRIVE >= 0.6.0, GEOS

This driver provides read access to road network elements of ASAM OpenDRIVE datasets.

ASAM OpenDRIVE is an open industry format for lane-detailed description of road networks, commonly used in applications of the automotive and transportation systems domains. It bundles polynomial, continuous road geometry modelling with all necessary topological links and semantic information from traffic-regulating infrastructure (signs and traffic lights).

7.87.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

7.87.2 Specification version

The currently supported OpenDRIVE version is 1.4 and depends on what is provided by libOpenDRIVE.

7.87.2.1 Supported OpenDRIVE elements

The XODR driver exposes OpenDRIVE's different road elements as separate layers by converting geometric elements into 3-dimensional OGR geometry types. The following layer types are currently implemented:

- *ReferenceLine*: Road reference line (`<planView>`) as *`OGRLineString`*.
- *LaneBorder*: Outer road lane border as *`OGRLineString`*.
- *Lane*: Polygonal surface (TIN) of the lane mesh as *`OGRTriangulatedSurface`*.
- *RoadMark*: Polygonal surface (TIN) of the road mark mesh as *`OGRTriangulatedSurface`*.
- *RoadObject*: Polygonal surface (TIN) of the road object mesh as *`OGRTriangulatedSurface`*.
- *RoadSignal*: Polygonal surface (TIN) of the road signal mesh as *`OGRTriangulatedSurface`*.

7.87.3 Spatial reference

By definition, OpenDRIVE geometries are always referenced in a Cartesian coordinate system which defaults to having its origin at `(0, 0, 0)`. If real-world coordinates are used, the OpenDRIVE header provides a PROJ.4 string with the definition of a projected spatial reference system:

```
<header ...>
  <geoReference><![CDATA[+proj=tmerc +lat_0=0 +lon_0=9 +k=0.9996 +x_0=500000 +y_0=0␣
↪+datum=WGS84 +units=m +no_defs]]></geoReference>
</header>
```

The XODR driver uses this PROJ definition as spatial reference for creation of all OGR geometry layers.

7.87.4 Limitations

The supported content encoding of OpenDRIVE XML files is limited to what pugixml is able to automatically guess (see 4.6. Encodings). The default fallback encoding is UTF-8.

7.87.5 Open options

The following open options can be specified (typically with the `-oo name=value` parameters of **`ogrinfo`** or **`ogr2ogr`**):

- **EPSILON=<float>:** Defaults to `1.0`. Epsilon value > `0.0` for linear approximation of continuous OpenDRIVE geometries. A smaller value results in a finer sampling. This parameter corresponds to libOpenDRIVE's `eps` parameter.
- **DISSOLVE_TIN=[YES/NO]:** Defaults to `NO`. Whether to dissolve triangulated surfaces. By setting this option to YES, the TIN layers *Lane* and *RoadMark* of geometry type *`OGRTriangulatedSurface`* will be simplified to single, simple *`OGRPolygon`* geometries. This performs a *`OGRGeometry::UnaryUnion()`* which dissolves boundaries of all touching triangle patches and thus yields a slimmer dataset which often suffices for basic GIS

usage. Be aware that this dissolving step increases processing time significantly. Layer *RoadSignal* will be dissolved to a simple `OGRPoint`.

7.87.6 Examples

7.87.6.1 Example 1: Translating OpenDRIVE road ReferenceLine elements (<planView>) to Shapefile using ogr2ogr.

The desired *layer type* which is to be extracted from the dataset is specified as the last parameter of the function call.

```
ogr2ogr -f "ESRI Shapefile" CulDeSac.shp CulDeSac.xodr ReferenceLine
```

7.87.6.2 Example 2: Converting the whole OpenDRIVE dataset with all its different layers into a GeoPackage

```
ogr2ogr -f "GPKG" CulDeSac.gpkg CulDeSac.xodr
```

7.87.6.3 Example 3: Converting the whole OpenDRIVE dataset with custom parameters EPSILON and DISSOLVE_TIN into a GeoPackage:

```
ogr2ogr -f "GPKG" CulDeSac.gpkg CulDeSac.xodr -oo EPSILON=0.9 -oo DISSOLVE_TIN=YES
```

7.87.7 Convenient usage through docker image

To use the XODR driver through a docker image, first build the image from the corresponding docker directory

```
cd <gdal>/docker/ubuntu-full/
docker build -t gdal/xodr -f Dockerfile .
```

For general usage information refer to GDAL Docker images. Usage examples:

- Use **ogrinfo** to extract detailed information about a local *xodr* file by mounting your current working directory (*$PWD*) containing the file into the Docker container:

```
docker run --rm -v ${PWD}:/home -it gdal/xodr ogrinfo /home/<file>.xodr
```

- Use **ogr2ogr** to convert a local *xodr* file into any other supported OGR output format. The result will be automatically available in your host machine's working directory which is mounted into the container:

```
docker run --rm -v ${PWD}:/home -it gdal/xodr ogr2ogr -f "GPKG" /home/<file>.gpkg /
↪home/<file>.xodr
```

Alternatively, you can run a docker container that enables using the XODR driver in an isolated workspace from within the container

```
docker run --name <container_name> -it gdal/xodr /bin/bash
```

7.87.8 General building notes

Building of the driver as plugin is tested to work on

- Ubuntu 24.04 using GCC
- Windows 10 using GCC 13.1.0 (with MCF threads) + MinGW-w64 11.0.0 (MSVCRT runtime), which is obtainable from WinLibs.

Ensure to meet the following driver dependencies:

- PROJ
- GEOS
- libOpenDRIVE as shared library (built with CMake option `-DBUILD_SHARED_LIBS=ON`)

Then, after checking out GDAL sources with this driver extension, create the build directory:

```
cd <gdal>
mkdir build
cd build
```

From the build directory configure CMake to activate our XODR driver as plugin:

```
cmake .. -DOGR_ENABLE_DRIVER_XODR_PLUGIN=TRUE -DOpenDrive_DIR=/path/to/
↪libOpenDRIVE/installdir/cmake/
```

Note

The `cmake/` path is usually automatically created when installing libOpenDRIVE and contains the necessary configuration files for inclusion into other project builds.

Now, build GDAL and install it:

```
cmake --build .
cmake --build . --target install
```

Afterwards you will find a new shared library file *path/to/GDAL/installdir*/lib/gdalplugins/ogr_XODR.

7.87.8.1 Verifying a successful build

Check if XODR driver is found:

```
cd <gdal>/build/
./apps/ogrinfo --format XODR
```

This should print basic capabilities of the driver:

```
Format Details:
  Short Name: XODR
  Long Name: OpenDRIVE - Open Dynamic Road Information for Vehicle Environment
  Supports: Vector
  Supports: Open() - Open existing dataset.
<OpenOptionList>
  ...
</OpenOptionList>
```

If you are on Linux, depending on your environment, you might experience linker errors like:

```
ERROR 1: libOpenDrive.so: cannot open shared object file: No such file or␣
↪directory
```

In such cases ensure that your environment variable `LD_LIBRARY_PATH` points to the corresponding install directories of libOpenDRIVE and GDAL and run `ldconfig` afterwards.

CHAPTER

EIGHT

USER ORIENTED DOCUMENTATION

8.1 GDAL Command-line Cookbook

Contents

- *GDAL Command-line Cookbook*
  - *Introduction*
  - *General How do I...*
  - *Raster How do I...*
  - *Vector How do I...*
  - *Raster and Vector How do I...*

8.1.1 Introduction

Welcome to the GDAL Users Cookbook. This guide provides practical examples for working with raster and vector data using the GDAL command-line tools. It demonstrates common tasks such as raster analysis, vector geometry operations, and combining raster and vector workflows.

The cookbook uses a question and answer format, linking to relevant examples elsewhere in the GDAL documentation.

8.1.2 General How do I...

... get the GDAL version?
Display the GDAL version

... view debug output?
Convert vector datasets with debug output

... write GDAL output and errors to a log file?
Log debug output and errors to a file

... log curl output for HTTP requests?
Get raster information with HTTP request logging enabled

8.1.3 Raster How do I...

... resize a raster?
Resize a dataset to 1000 columns and 500 lines using cubic resampling.

... create a Cloud Optimized GeoTIFF (COG)?

- *Converting file* `utm.tif` *to a cloud-optimized GeoTIFF using JPEG compression*
- *Create a COG with* `gdalwarp`

... check if a GeoTIFF is a Cloud Optimized GeoTIFF (COG)?

- *Check if a GeoTIFF file is cloud optimized using* `--detailed`
- *Check if a remote GeoTIFF is a COG*

8.1.4 Vector How do I...

... check if `X` vector driver is installed?
Check if the vector Parquet driver is installed

... buffer geometries?
Compute a buffer of one km around input geometries (assuming the CRS is in meters)

... buffer geometries using an attribute?
Buffer a line dataset to create a new polygon dataset.

... list all layers in a dataset?

- *List all layers in a dataset using* `jq`
- *List all layers and their geometry fields using* `jq`

... filter features using an attribute query?
Filter Shapefile features with an attribute query

... list unique values in a field?
List unique values in a Shapefile field

8.1.5 Raster and Vector How do I...

... burn a vector dataset into a raster?
Burn a shapefile into a raster

... extract pixel values from a raster and apply them to a point dataset?
Reading coordinates to extract from an input GeoPackage file and writing the output to a GeoPackage file

... know which CRSs are suitable for reprojecting my dataset?
Auto-completion of EPSG CRS codes

... clip a raster image with vector features?

- *Clip a GeoTIFF file using vector features*
- *Clip a GeoTIFF file using SQL*
- *Clip a raster dataset using a nested vector pipeline*
- *Clip a raster dataset with vector features using* `_PIPE_`

8.2 GDAL CLI Debug and Logging Output

Contents

- *GDAL CLI Debug and Logging Output*
  - *Debug Output*
    - *Example 1: Convert vector datasets with debug output*
    - *Example 2: Filter debug output by driver*
  - *Logging and Diagnostic Options*
    - *Example 3: Get raster information with HTTP request logging enabled*
    - *Example 4: Get raster info with curl logging and timestamps*
    - *Example 5: Log debug output and errors to a file*

8.2.1 Debug Output

To display GDAL debug output for a command, use the `--debug` option. See *Common options for raster programs* and *Common options for vector programs* for details on `--debug`. The following example enables debug output for *gdal vector info*.

```
$ gdal vector info in.gpkg --debug ON
```

When `debug` is enabled, GDAL writes additional messages to the command error stream. These messages can help identify issues related to dataset access, driver loading, configuration options, and processing operations.

Each debug message includes a prefix indicating the component that produced it. The prefixes typically indicate drivers (e.g. `Shape`, `GeoJSON`), tools (e.g. `GDALVectorTranslate`), or the core GDAL runtime (`GDAL`).

8.2.1.1 Example 1: Convert vector datasets with debug output

The following example shows debug output with different prefix types.

```
$ gdal vector convert in.shp out.geojson --overwrite --debug ON
GDAL: GDALOpen(in.shp, this=000001719D0A3D80) succeeds as ESRI Shapefile.
GeoJSON: First pass: 100.00 %
GDAL: GDALOpen(out.geojson, this=000001719D0AC240) succeeds as GeoJSON.
GDAL: GDALClose(out.geojson, this=000001719D0AC240)
GDAL: Using GeoJSON driver
GDAL: GDALDriver::Create(GeoJSON,out.geojson,0,0,0,Unknown,0000000000000000)
0...10...20...30...40...50...60...70...80...90...100 - done.
GDALVectorTranslate: 19 features written in layer 'in'
Shape: 19 features read on layer 'in'.
```

Debug output can be filtered by passing a component prefix to the `--debug` option. For example, to display only messages beginning with `Shape:`, use ``--debug Shape`. Prefix matching is case-insensitive, but partial matches and wildcards are not supported.

8.2.1.2 Example 2: Filter debug output by driver

```
$ gdal vector convert in.shp out.geojson --overwrite --debug Shape
0...10...20...30...40...50...60...70...80...90...100 - done.
Shape: 19 features read on layer 'in'.
```

Note

The `--debug` option accepts `ON` to enable debug output, `OFF` to disable it, or a prefix string to filter debug messages. If none of these values are provided, GDAL returns the following error:

```
ERROR 1: --debug option given without debug level.
```

8.2.2 Logging and Diagnostic Options

There are several configuration options that control GDAL logging output. These are documented in the *Logging* section of the config options page.

The options are prefixed with CPL ("Common Portability Library"), the internal GDAL utility library used by GDAL core and drivers.

Options can be set per command using the `--config` option, or globally using environment variables. Note that `--debug ON` is equivalent to `--config CPL_DEBUG=ON`.

The following are all broadly equivalent, although environment variables affect all subsequent GDAL commands in the same session, whereas `--config` only applies to the current command.

Bash

PowerShell

```
$ gdal vector info in.gpkg --debug ON
$ gdal vector info in.gpkg --config CPL_DEBUG=ON

# set CPL_DEBUG for the scope of the single command
$ CPL_DEBUG=ON gdal vector info in.gpkg

# set CPL_DEBUG for the whole shell session
$ export CPL_DEBUG=ON
$ gdal vector info in.gpkg
```

```
gdal vector info in.gpkg --debug ON
gdal vector info in.gpkg --config CPL_DEBUG=ON

# set CPL_DEBUG for the whole shell session
$env:CPL_DEBUG="ON"
gdal vector info in.gpkg
```

When working with remote datasets, it is often useful to log curl output for HTTP requests. This can be done by setting the *CPL_CURL_VERBOSE* configuration option to `ON`.

8.2.2.1 Example 3: Get raster information with HTTP request logging enabled

```
$ gdal raster info /vsicurl/https://download.osgeo.org/geotiff/samples/spot/chicago/
↪SP27GTIF.TIF --config CPL_CURL_VERBOSE=ON
```

CPL_TIMESTAMP adds timestamps to each log and debug message emitted by GDAL, which can be useful for troubleshooting performance bottlenecks and benchmarking.

8.2.2.2 Example 4: Get raster info with curl logging and timestamps

Bash

PowerShell

```
$ export CPL_TIMESTAMP=ON
$ gdal raster info /vsicurl/https://download.osgeo.org/geotiff/samples/spot/chicago/
↪SP27GTIF.TIF --debug "CURL_INFO_TEXT" --config CPL_CURL_VERBOSE=ON
[Fri May 15 11:57:24 2026].6370, 0.0000: CURL_INFO_TEXT: Could not find host download.
↪osgeo.org in the .netrc file; using defaults
[Fri May 15 11:57:24 2026].6640, 0.0270: CURL_INFO_TEXT: Host download.osgeo.org:443 was␣
↪resolved.
```

```
$env:CPL_TIMESTAMP="ON"
gdal raster info /vsicurl/https://download.osgeo.org/geotiff/samples/spot/chicago/
↪SP27GTIF.TIF --debug "CURL_INFO_TEXT" --config CPL_CURL_VERBOSE=ON
```

CPL_LOG_ERRORS enables logging of error messages through GDAL's CPL logging system, rather than only reporting them to the terminal. When combined with *CPL_LOG*, log output (including errors) can be written to a file.

8.2.2.3 Example 5: Log debug output and errors to a file

Bash

PowerShell

```
export CPL_DEBUG=ON
export CPL_LOG_ERRORS=ON
export CPL_TIMESTAMP=ON
gdal raster info /vsicurl/https://download.osgeo.org/geotiff/samples/spot/chicago/
↪SP27GTIF.TIF --config CPL_LOG=/tmp/gdal.log
```

```
$env:CPL_DEBUG="ON"
$env:CPL_LOG_ERRORS="ON"
$env:CPL_TIMESTAMP="ON"
gdal raster info /vsicurl/https://download.osgeo.org/geotiff/samples/spot/chicago/
↪SP27GTIF.TIF --config CPL_LOG=C:\Temp\gdal.log
```

8.3 Error Codes

8.3.1 Overview

This page describes GDAL error codes that may appear in command-line error output or be reported by the Python API. Each GDAL error code is listed below with:

- a brief description,

- examples of operations that may produce the error, and
- possible resolutions

GDAL may also report warnings, which indicate potential issues that do not prevent the operation from completing. These use the same codes as errors. The different levels are listed at *Error levels*. See *Error handling in GDAL* for details on GDAL error handling using the C and C++ APIs, and more details on error codes and levels.

Contents

- *Error Codes*
  - *Overview*
  - *General Error Codes*
    - *ERROR 1 Application Defined Error*
      - *Example 1: Overwriting a GeoPackage that is in use by another application*
      - *Example 2: Creating GeoTIFFs larger than 4 GB*
      - *Example 3: Converting a Shapefile with multipolygons to a GeoPackage*
      - *Example 4: Converting to a parquet file*
    - *ERROR 2 Out of Memory Error*
    - *ERROR 3 File I/O Error*
      - *Example 5: Not enough disk space*
    - *ERROR 4 Open Failed*
      - *Example 6: Opening a NetCDF file*
      - *Example 7: Opening a zipped Shapefile*
    - *ERROR 5 Illegal Argument*
      - *Example 8: Using an unknown command-line option*
    - *ERROR 6 Not Supported*
      - *Example 9: JPEG compression fails on a 4-band GeoTIFF*
    - *ERROR 7 Assertion Failed*
    - *ERROR 8 No Write Access*
    - *ERROR 9 User Interrupted*
  - *Network Related Error Codes*
    - *ERROR 10 HTTP Response*
    - *ERROR 11 Bucket Not Found*
    - *ERROR 12 Object Not Found*
    - *ERROR 13 Access Denied*
    - *ERROR 14 Invalid Credentials*
    - *ERROR 15 Signature Does Not Match*
    - *ERROR 16 Object Storage Generic Error*

8.3.2 General Error Codes

8.3.2.1 ERROR 1 Application Defined Error

Code: `CPLE_AppDefined`

A generic error indicating that the application, driver, or GDAL component reported a failure that does not match a more specific error classification. Additional details are typically provided in the accompanying error message.

Example 1: Overwriting a GeoPackage that is in use by another application

Although `--overwrite` is specified, the output file cannot be replaced because it is in use by another application (for example, when opened in QGIS on Windows).

```
$ gdal vector buffer --distance=5 in.gpkg out.gpkg --overwrite
ERROR 1: A file system object called 'out.gpkg' already exists.
ERROR 1: GPKG driver failed to create out.gpkg
```

Example 2: Creating GeoTIFFs larger than 4 GB

This error occurs when writing a TIFF that exceeds the standard 4 GB file size limit. To resolve it, enable the BigTIFF format using the `BIGTIFF=YES` creation option, as indicated in the error message.

```
$ gdal raster mosaic --output-format COG *.tif out.tif
ERROR 1: TIFFAppendToStrip: Maximum TIFF file size exceeded. Use BIGTIFF=YES creation␣
↪option.
```

Example 3: Converting a Shapefile with multipolygons to a GeoPackage

Shapefiles do not clearly distinguish between Polygon and MultiPolygon geometries. For historical reasons, the driver always reports the layer geometry type as `POLYGON`, even when some or all features are `MULTIPOLYGON` geometries.

As a result, the conversion succeeds but emits a warning, and the resulting GeoPackage is not fully compliant with the specification.

To create a compliant GeoPackage, explicitly set the output layer geometry type to `MULTIPOLYGON`. See *Convert a shapefile mixing polygons and multipolygons to a GeoPackage with multipolygons*.

```
$ gdal convert poly_multipoly.shp out.gpkg
Warning 1: A geometry of type MULTIPOLYGON is inserted into layer poly_multipoly of␣
↪geometry type POLYGON,
which is not normally allowed by the GeoPackage specification, but the driver will␣
↪however do it.
To create a conformant GeoPackage, if using ogr2ogr, the -nlt option can be used to␣
↪override the layer geometry type.
This warning will no longer be emitted for this combination of layer and feature␣
↪geometry type.
```

Example 4: Converting to a parquet file

GDAL supports *(Geo)Parquet*, but in some installations the Parquet driver is provided as a separate plugin.

In a *Conda*-based GDAL environment, the missing plugin can be installed with `conda install`.

```
$ gdal vector convert in.gpkg out.parquet
ERROR 1: No installed driver matching parquet extension, but Parquet driver is known.
However plugin ogr_Parquet.dll is not available in your installation.You may install it␣
↪with
'conda install -c gdal-master libgdal-arrow-parquet'
```

8.3.2.2 ERROR 2 Out of Memory Error

Code: `CPLE_OutOfMemory`

An error indicating that GDAL or a driver was unable to allocate the memory required to complete the operation. This typically occurs when processing large datasets, performing memory-intensive transformations, or when system memory limits have been reached.

8.3.2.3 ERROR 3 File I/O Error

Code: `CPLE_FileIO`

An error indicating a failure while reading from or writing to a file or other data source. This may occur due to issues such as missing files, permission problems, corrupted storage, interruptions during disk or network operations, or failures when accessing remote datasets (e.g., /vsicurl streams).

Example 5: Not enough disk space

In this example, there is not enough free space available on the disk to store the output dataset. Freeing up disk space will resolve the error.

Alternatively, using a smaller output format can reduce disk usage. For example, `--output-format COG` typically produces significantly smaller files (because it defaults to using LZW compression).

> Finally, as suggested in the error message, the *GTiff -- GeoTIFF File Format* creation option `CHECK_DISK_FREE_SPACE=FALSE` can be used to disable the free disk space check. This is typically only required when writing compressed GeoTIFF files (for example with `COMPRESS=DEFLATE`), since GDAL cannot predict the final compressed file size exactly and instead uses heuristics to estimate the required disk space.

```
$ gdal raster calc -i "A=a.tif" -i "B=b.tif" --calc "A - 0.3*B" --nodata=-9999 -o c.tif
ERROR 3: c.tif: Free disk space available is 1.62 GB, whereas 46.57 GB are at least␣
↪necessary.
You can disable this check by defining the CHECK_DISK_FREE_SPACE configuration option to␣
↪FALSE.
```

8.3.2.4 ERROR 4 Open Failed

Code: `CPLE_OpenFailed`

An error indicating that GDAL was unable to open a dataset or file. This may occur if the file does not exist, is inaccessible due to permissions, is corrupted, or is not recognised as a supported format by any available driver.

To check if a driver is installed, see *Check if the vector Parquet driver is installed*.

Example 6: Opening a NetCDF file

GDAL supports *NetCDF: Network Common Data Form*, but some packaging systems require the netCDF driver to be installed separately.

As indicated in the error message, this can be installed in a *Conda*-based GDAL environment using `conda install`.

```
$ gdal vector info trmm-2x2.nc
ERROR 4: `trmm-2x2.nc` not recognized as being in a supported file format.
It could have been recognized by driver netCDF, but plugin gdal_netCDF.dll is not␣
↪available in your installation.
You may install it with 'conda install -c conda-forge libgdal-netcdf'
```

Example 7: Opening a zipped Shapefile

Even though the ESRI Shapefile driver supports compressed archives (`*.zip`), attempting to open a ZIP file directly may result in an error.

In this case, the dataset is stored inside a ZIP archive. GDAL requires the */vsizip/ (.zip archives)* virtual filesystem prefix to access its contents.

```
$ gdal vector info C:\Data\in.zip
ERROR 4: 'C:\Data\in.zip' not recognized as being in a supported file format.
Changing the filename to /vsizip/C:\Data\in.zip may help it to be recognized.

$ gdal vector info "/vsizip/C:\Data\in.zip"
INFO: Open of '/vsizip/C:\Data\in.zip'
  using driver 'ESRI Shapefile' successful.
```

8.3.2.5 ERROR 5 Illegal Argument

Code: `CPLE_IllegalArg`

An invalid or unsupported argument was passed to a GDAL function or command.

Example 8: Using an unknown command-line option

The `gdal raster reproject` command does not support the `--crs` option. Use `--output-crs` instead. Available options can be listed using `--help` or by checking the documentation for *gdal raster reproject*.

```
$ gdal raster reproject --crs=EPSG:32632 in.tif out.tif --overwrite
ERROR 5: reproject: Option '--crs' is unknown.
Usage: gdal raster reproject [OPTIONS] <INPUT> <OUTPUT>
Try 'gdal raster reproject --help' for help.
```

8.3.2.6 ERROR 6 Not Supported

Code: `CPLE_NotSupported`

The requested operation is not supported by the driver, dataset, or GDAL component.

Example 9: JPEG compression fails on a 4-band GeoTIFF

JPEG compression with `PHOTOMETRIC=YCBCR` is only supported for 3-band RGB datasets. To create a compatible dataset, select 3 bands from the source image before conversion. See *Convert a RGBA dataset to a YCbCR JPEG compressed GeoTIFF*.

See *GTiff -- GeoTIFF File Format* for supported combinations of compression methods and photometric interpretations.

```
$ gdal raster convert --co COMPRESS=JPEG --co PHOTOMETRIC=YCBCR rgba.tif output.tif
ERROR 6: output.tif: PHOTOMETRIC=YCBCR not supported on a 4-band raster:
only compatible of a 3-band (RGB) raster
```

8.3.2.7 ERROR 7 Assertion Failed

Code: `CPLE_AssertionFailed`

An internal GDAL assertion failed, typically indicating a programming or logic error.

8.3.2.8 ERROR 8 No Write Access

Code: `CPLE_NoWriteAccess`

Write access to the dataset, file, or resource was denied.

8.3.2.9 ERROR 9 User Interrupted

Code: `CPLE_UserInterrupt`

The operation was cancelled or interrupted by the user.

8.3.3 Network Related Error Codes

8.3.3.1 ERROR 10 HTTP Response

Code: `CPLE_HttpResponse`

The server returned an error response to an HTTP request. This typically occurs when accessing remote datasets over HTTP or HTTPS.

8.3.3.2 ERROR 11 Bucket Not Found

Code: `CPLE_BucketNotFound`

The requested object storage bucket could not be found.

```
ERROR 11:  HTTP response code:  404
```

8.3.3.3 ERROR 12 Object Not Found

Code: `CPLE_ObjectNotFound`

The requested object (such as a file or resource) does not exist in the remote storage system.

8.3.3.4 ERROR 13 Access Denied

Code: `CPLE_AccessDenied`

Access to the requested resource was denied by the remote storage system.

8.3.3.5 ERROR 14 Invalid Credentials

Code: `CPLE_InvalidCredentials`

Authentication failed due to invalid or missing credentials.

8.3.3.6 ERROR 15 Signature Does Not Match

Code: `CPLE_SignatureDoesNotMatch`

The provided authentication signature does not match the expected value.

8.3.3.7 ERROR 16 Object Storage Generic Error

Code: `CPLE_ObjectStorageGenericError`

A generic object storage error that does not fit a more specific error category.

8.4 Raster Data Model

This document attempts to describe the GDAL data model. That is the types of information that a GDAL data store can contain, and their semantics.

8.4.1 Dataset

A dataset (represented by the *`GDALDataset`* class) is an assembly of related raster bands and some information common to them all. In particular the dataset has a concept of the raster size (in pixels and lines) that applies to all the bands. The dataset is also responsible for the georeferencing transform and coordinate system definition of all bands. The dataset itself can also have associated metadata, a list of name/value pairs in string form.

The number of pixels and lines for a raster band is limited to 2,147,483,647 each. The number of bands is also limited to 2,147,483,647, although by default a limitation to 65,536 is applied to avoid excessive RAM consumption.

Note that the GDAL dataset, and raster band data model is loosely based on the OpenGIS Grid Coverages specification.

8.4.2 Coordinate System

Dataset coordinate systems are represented as OpenGIS Well Known Text strings. This can contain:

- An overall coordinate system name.
- A geographic coordinate system name.
- A datum identifier.
- An ellipsoid name, semi-major axis, and inverse flattening.
- A prime meridian name and offset from Greenwich.
- A projection method type (e.g., Transverse Mercator).
- A list of projection parameters (e.g., central_meridian).
- A units name, and conversion factor to meters or radians.
- Names and ordering for the axes.
- Codes for most of the above in terms of predefined coordinate systems from authorities such as EPSG.

For more information on OpenGIS WKT coordinate system definitions, and mechanisms to manipulate them, refer to the osr_tutorial document and/or the OGRSpatialReference class documentation.

The coordinate system returned by *`GDALDataset::GetProjectionRef()`* describes the georeferenced coordinates implied by the affine georeferencing transform returned by *`GDALDataset::GetGeoTransform()`*. The coordinate system returned by *`GDALDataset::GetGCPProjection()`* describes the georeferenced coordinates of the GCPs returned by *`GDALDataset::GetGCPs()`*.

Note that a returned coordinate system strings of "" indicates nothing is known about the georeferencing coordinate system.

8.4.3 Affine GeoTransform

GDAL datasets have two ways of describing the relationship between raster positions (in pixel/line coordinates) and georeferenced coordinates. The first, and most commonly used is the affine transform (the other is GCPs).

The affine transform consists of six coefficients returned by *GDALDataset::GetGeoTransform()* which map pixel/line coordinates into georeferenced space using the following relationship:

```
Xgeo = GT(0) + Xpixel*GT(1) + Yline*GT(2)
Ygeo = GT(3) + Xpixel*GT(4) + Yline*GT(5)
```

In case of north up images, the GT(2) and GT(4) coefficients are zero, and the GT(1) is pixel width, and GT(5) is pixel height. The (GT(0),GT(3)) position is the top left corner of the top left pixel of the raster.

Note that the pixel/line coordinates in the above are from (0.0,0.0) at the top left corner of the top left pixel to (width_in_pixels,height_in_pixels) at the bottom right corner of the bottom right pixel. The pixel/line location of the center of the top left pixel would therefore be (0.5,0.5).

8.4.4 GCPs

A dataset can have a set of control points relating one or more positions on the raster to georeferenced coordinates. All GCPs share a georeferencing coordinate system (returned by *GDALDataset::GetGCPProjection()*). Each GCP (represented as the GDAL_GCP class) contains the following:

```
typedef struct
{
    char        *pszId;
    char        *pszInfo;
    double      dfGCPPixel;
    double      dfGCPLine;
    double      dfGCPX;
    double      dfGCPY;
    double      dfGCPZ;
} GDAL_GCP;
```

The pszId string is intended to be a unique (and often, but not always numerical) identifier for the GCP within the set of GCPs on this dataset. The pszInfo is usually an empty string, but can contain any user defined text associated with the GCP. Potentially this can also contain machine parsable information on GCP status though that isn't done at this time.

The (Pixel,Line) position is the GCP location on the raster. The (X,Y,Z) position is the associated georeferenced location with the Z often being zero.

The GDAL data model does not imply a transformation mechanism that must be generated from the GCPs ... this is left to the application. However 1st to 5th order polynomials are common.

Normally a dataset will contain either an affine geotransform, GCPs or neither. It is uncommon to have both, and it is undefined which is authoritative.

8.4.5 Metadata

GDAL metadata is auxiliary format and application specific textual data kept as a list of name/value pairs. The names are required to be well behaved tokens (no spaces, or odd characters). The values can be of any length, and contain anything except an embedded null (ASCII zero).

The metadata handling system is not well tuned to handling very large bodies of metadata. Handling of more than 100K of metadata for a dataset is likely to lead to performance degradation.

Some formats will support generic (user defined) metadata, while other format drivers will map specific format fields to metadata names. For instance the TIFF driver returns a few information tags as metadata including the date/time field which is returned as:

```
TIFFTAG_DATETIME=1999:05:11 11:29:56
```

Metadata is split into named groups called domains, with the default domain having no name (NULL or ""). Some specific domains exist for special purposes. Note that currently there is no way to enumerate all the domains available for a given object, but applications can "test" for any domains they know how to interpret.

The following metadata items have well defined semantics in the default domain:

- AREA_OR_POINT: May be either "Area" (the default) or "Point". Indicates whether a pixel value should be assumed to represent a sampling over the region of the pixel or a point sample at the center of the pixel. This is not intended to influence interpretation of georeferencing which remains area oriented.
- NODATA_VALUES: The value is a list of space separated pixel values matching the number of bands in the dataset that can be collectively used to identify pixels that are nodata in the dataset. With this style of nodata a pixel is considered nodata in all bands if and only if all bands match the corresponding value in the NODATA_VALUES tuple. This metadata is not widely honoured by GDAL drivers, algorithms or utilities at this time.
- MATRIX_REPRESENTATION: This value, used for Polarimetric SAR datasets, contains the matrix representation that this data is provided in. The following are acceptable values:
  - SCATTERING
  - SYMMETRIZED_SCATTERING
  - COVARIANCE
  - SYMMETRIZED_COVARIANCE
  - COHERENCY
  - SYMMETRIZED_COHERENCY
  - KENNAUGH
  - SYMMETRIZED_KENNAUGH
- POLARIMETRIC_INTERP: This metadata item is defined for Raster Bands for polarimetric SAR data. This indicates which entry in the specified matrix representation of the data this band represents. For a dataset provided as a scattering matrix, for example, acceptable values for this metadata item are HH, HV, VH, VV. When the dataset is a covariance matrix, for example, this metadata item will be one of Covariance_11, Covariance_22, Covariance_33, Covariance_12, Covariance_13, Covariance_23 (since the matrix itself is a hermitian matrix, that is all the data that is required to describe the matrix).
- METADATATYPE: If IMAGERY Domain present, the item consist the reader which processed the metadata. Now present such readers:
  - DG: DigitalGlobe imagery metadata
  - GE: GeoEye (or formally SpaceImaging) imagery metadata
  - OV: OrbView imagery metadata
  - DIMAP: Pleiades imagery metadata
  - MSP: Resurs DK-1 imagery metadata
  - ODL: Landsat imagery metadata
- CACHE_PATH: A cache directory path. Now this metadata item sets only by WMS driver. This is useful when dataset deletes with cached data or when external program need to put tiles in cache for some area of interest.

8.4.5.1 SUBDATASETS Domain

The SUBDATASETS domain holds a list of child datasets. Normally this is used to provide pointers to a list of images stored within a single multi image file.

For example, an NITF with two images might have the following subdataset list.

```
SUBDATASET_1_NAME=NITF_IM:0:multi_1b.ntf
SUBDATASET_1_DESC=Image 1 of multi_1b.ntf
SUBDATASET_2_NAME=NITF_IM:1:multi_1b.ntf
SUBDATASET_2_DESC=Image 2 of multi_1b.ntf
```

The value of the _NAME is the string that can be passed to *GDALOpen()* to access the file. The _DESC value is intended to be a more user friendly string that can be displayed to the user in a selector.

Drivers which support subdatasets advertise the `DMD_SUBDATASETS` capability. This information is reported when the --format and --formats options are passed to the command line utilities.

Currently, drivers which support subdatasets are: ADRG, ECRGTOC, GEORASTER, GTiff, HDF4, HDF5, netCDF, NITF, NTv2, OGDI, PDF, PostGISRaster, Rasterlite, RPFTOC, RS2, TileDB, WCS, WMS, WMTS, and Zarr.

8.4.5.2 IMAGE_STRUCTURE Domain

Metadata in the default domain is intended to be related to the image, and not particularly related to the way the image is stored on disk. That is, it is suitable for copying with the dataset when it is copied to a new format. Some information of interest is closely tied to a particular file format and storage mechanism. In order to prevent this getting copied along with datasets it is placed in a special domain called IMAGE_STRUCTURE that should not normally be copied to new formats.

Currently the following items are defined by rfc-14 as having specific semantics in the IMAGE_STRUCTURE domain.

- COMPRESSION: The compression type used for this dataset or band. There is no fixed catalog of compression type names, but where a given format includes a COMPRESSION creation option, the same list of values should be used here as there.
- NBITS: The actual number of bits used for this band, or the bands of this dataset. Normally only present when the number of bits is non-standard for the datatype, such as when a 1 bit TIFF is represented through GDAL as GDT_UInt8.
- INTERLEAVE: This only applies on datasets, and the value should be one of PIXEL, LINE or BAND. It can be used as a data access hint. See *Multiband pixel organization (INTERLEAVE metadata item)* for more details.
- PIXELTYPE: This may appear on a GDT_UInt8 band (or the corresponding dataset) and have the value SIGNEDBYTE to indicate the unsigned byte values between 128 and 255 should be interpreted as being values between -128 and -1 for applications that recognise the SIGNEDBYTE type. Starting with GDAL 3.7, this metadata item is no longer used, as the Int8 data type is used for signed bytes.

Multiband pixel organization (INTERLEAVE metadata item)

For a dataset with several channel, there are 3 typical ways of how to organize image data within the raster file.

- In Band Interleaved by Pixel (BIP) mode, reported as INTERLEAVE=PIXEL, the samples of all bands for a given pixel are stored contiguously. Said otherwise, for a given (row, column) location in the raster, the pixel value of the first band is first written, followed by the value of the second band, until the value of the last band.

  For example, for a 3-band image the values are stored as `R1 G1 B1 R2 G2 B2 R3 G3 B3`, where `R1 G1 B1` are the band values of the first pixel, `R2 G2 B2` those of the second pixel, and so on.

  That mode is efficient to fetch at once the values of all bands for a given location, and is conversely less efficient to extract only a subset of bands when the number of bands increases.

Fig. 1: Band Interleaved by Pixel (BIP) mode

In GeoTIFF format, this organization corresponds to the PlanarConfiguration=Contiguous, and there is only one single tile or strip (depending on whether the dataset is tiled or not) for each spatial region. That mode can be required for some compression methods, or be preferred for higher compression, such as using YCbCr photometric mode for JPEG compression.

- In Band SeQuential (BSQ) mode, reported as INTERLEAVE=BAND, all samples of one band are stored together before the next band.

  For example, for a 3-band image the values are stored as `R1 R2 R3 ... G1 G2 G3 ... B1 B2 B3 ...`, where all pixels of the first band are stored first, followed by all pixels of the second band, and then the third band.

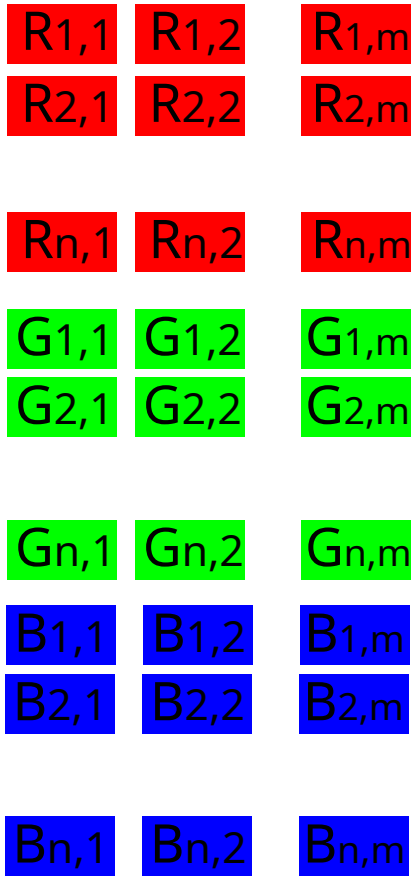


Fig. 2: Band SeQuential (BSQ) mode

In GeoTIFF format, this organization corresponds to the PlanarConfiguration=Separate, and there is one tile or strip (depending on whether the dataset is tiled or not) per band for each spatial region.

- In Band Interleaved by Line (BIL) mode, reported as INTERLEAVE=LINE, the samples are stored band by band for each image line.

  For a 3-band image, the layout for each line is:

```
R11 R21 R31 ... G11 G21 G31 ... B11 B21 B31 ...  (line 1)
R21 R22 R32 ... G21 G22 G23 ... B21 B22 B32 ...  (line 2)
```

  That is, all pixels of band 1 for the first line are stored first, followed by all pixels of band 2 for that line, then band 3, before moving to the next line.

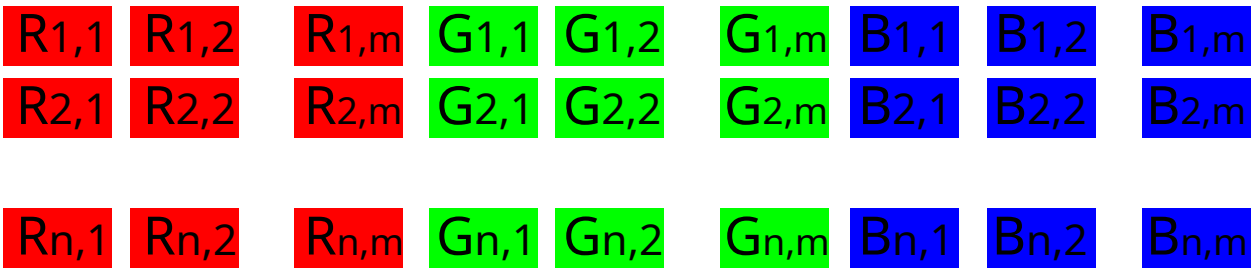


Fig. 3: Band Interleaved by Line (BIL) mode

  There is no GeoTIFF pixel organization corresponding to that mode, but it can be found in some raw image formats such as *ESRI .hdr labeling* or *ENVI .hdr Labelled Raster*

8.4.5.3 RPC Domain

The RPC metadata domain holds metadata describing the Rational Polynomial Coefficient geometry model for the image if present. This geometry model can be used to transform between pixel/line and georeferenced locations. The items defining the model are:

- ERR_BIAS: Error - Bias. The RMS bias error in meters per horizontal axis of all points in the image (-1.0 if unknown)
- ERR_RAND: Error - Random. RMS random error in meters per horizontal axis of each point in the image (-1.0 if unknown)
- LINE_OFF: Line Offset
- SAMP_OFF: Sample Offset
- LAT_OFF: Geodetic Latitude Offset
- LONG_OFF: Geodetic Longitude Offset
- HEIGHT_OFF: Geodetic Height Offset
- LINE_SCALE: Line Scale
- SAMP_SCALE: Sample Scale
- LAT_SCALE: Geodetic Latitude Scale
- LONG_SCALE: Geodetic Longitude Scale
- HEIGHT_SCALE: Geodetic Height Scale
- LINE_NUM_COEFF (1-20): Line Numerator Coefficients. Twenty coefficients for the polynomial in the Numerator of the rn equation. (space separated)

- LINE_DEN_COEFF (1-20): Line Denominator Coefficients. Twenty coefficients for the polynomial in the Denominator of the rn equation. (space separated)
- SAMP_NUM_COEFF (1-20): Sample Numerator Coefficients. Twenty coefficients for the polynomial in the Numerator of the cn equation. (space separated)
- SAMP_DEN_COEFF (1-20): Sample Denominator Coefficients. Twenty coefficients for the polynomial in the Denominator of the cn equation. (space separated)

These fields are directly derived from the document prospective GeoTIFF RPC document (http://geotiff.maptools.org/rpc_prop.html) which in turn is closely modeled on the NITF RPC00B definition.

The line and pixel offset expressed with LINE_OFF and SAMP_OFF are with respect to the center of the pixel.

8.4.5.4 IMAGERY Domain (remote sensing)

For satellite or aerial imagery the IMAGERY Domain may be present. It depends on the existence of special metadata files near the image file. The files at the same directory with image file tested by the set of metadata readers, if files can be processed by the metadata reader, it fill the IMAGERY Domain with the following items:

- SATELLITEID: A satellite or scanner name
- CLOUDCOVER: Cloud coverage. The value between 0 - 100 or 999 if not available
- ACQUISITIONDATETIME: The image acquisition date time in UTC

Starting with GDAL 3.10, there also exists a raster band level IMAGERY metadata domain with the following items:

- CENTRAL_WAVELENGTH_UM: Central Wavelength in micrometers.
- FWHM_UM: Full-width half-maximum (FWHM) in micrometers.

Clients can get (resp. set) these metadata items with *GDALRasterBand::GetMetadataItem()* (resp. *GDALRasterBand::SetMetadataItem()*).`

They are specifically set by the *SENTINEL2 -- Sentinel-2 Products* and *ENVI -- ENVI .hdr Labelled Raster* drivers (if corresponding metadata items are found in the ENVI header), but may also be found in other drivers handling arbitrary GDAL metadata, such as the one using the GDAL Persistent Auxiliary Mechanism (PAM / .aux.xml side car files) or *VRT -- GDAL Virtual Format* drivers. The *GTiff -- GeoTIFF File Format* driver also supports serializing and deserializing the band IMAGERY metadata domain in the `GDAL_METADATA` TIFF tag.

8.4.5.5 xml: Domains

Any domain name prefixed with "xml:" is not normal name/value metadata. It is a single XML document stored in one big string.

8.4.6 Raster Band

A raster band is represented in GDAL with the *GDALRasterBand* class. It represents a single raster band/channel/layer. It does not necessarily represent a whole image. For instance, a 24bit RGB image would normally be represented as a dataset with three bands, one for red, one for green and one for blue.

A raster band has the following properties:

- A width and height in pixels and lines. This is the same as that defined for the dataset, if this is a full resolution band.
- A datatype (GDALDataType). One of UInt8 (since 3.13, also available as Byte in all versions), Int8, UInt16, Int16, UInt32, Int32, UInt64, Int64, Float16, Float32, Float64, and the complex types CInt16, CInt32, CFloat16, CFloat32, and CFloat64.

  UInt64 and Int64 data types have been added in GDAL 3.5. Beyond reading and write pixel values, their support is limited. Some algorithms might use 64-bit floating-point internally (warping), as well as some methods returning

only double values (GetMinimum(), GetMaximum(), etc.), or even 32-bit floating point (overview, RasterIO resampling). Hence the range where exact values are preserved can be [0, 2^53] (or less if 32-bit floating-point is used).

The Int8 data type has been added in GDAL 3.7.

The Float16 and CFloat16 data types have been added in GDAL 3.11. If this data type is not supported by the hardware, then a software emulation is used. Not all drivers support Float16 yet.

- A block size. This is a preferred (efficient) access chunk size. For tiled images this will be one tile. For scanline oriented images this will normally be one scanline.
- A list of name/value pair metadata in the same format as the dataset, but of information that is potentially specific to this band.
- An optional description string.
- An optional single nodata pixel value (see also NODATA_VALUES metadata on the dataset for multi-band style nodata values).
- An optional nodata mask band marking pixels as nodata or in some cases transparency as discussed in RFC 15: Band Masks and documented in GDALRasterBand::GetMaskBand().
- An optional list of category names (effectively class names in a thematic image).
- An optional minimum and maximum value.
- Optional statistics stored in metadata:
  - STATISTICS_MEAN: mean
  - STATISTICS_MINIMUM: minimum
  - STATISTICS_MAXIMUM: maximum
  - STATISTICS_STDDEV: standard deviation
  - STATISTICS_APPROXIMATE: only present if GDAL has computed approximate statistics
  - STATISTICS_VALID_PERCENT: percentage of valid (not nodata) pixel
- An optional offset and scale for transforming raster values into meaning full values (e.g., translate height to meters).
- An optional raster unit name. For instance, this might indicate linear units for elevation data.
- A color interpretation for the band. This is one of:
  - GCI_Undefined / "Undefined": default, nothing is known.
  - GCI_GrayIndex / "Gray": independent gray-scale image
  - GCI_PaletteIndex / "Palette": this raster acts as an index into a color table
  - GCI_RedBand / "Red": red portion of an RGB or RGBA image, or red spectral band [0.62 - 0.69 um]
  - GCI_GreenBand/ "Green": green portion of an RGB or RGBA image, or green spectral band [0.51 - 0.60 um]
  - GCI_BlueBand / "Blue": blue portion of an RGB or RGBA image, or blue spectral band [0.45 - 0.53 um]
  - GCI_AlphaBand / "Alpha": alpha portion of an RGBA image
  - GCI_HueBand / "Hue": hue of a HLS image
  - GCI_SaturationBand / "Saturation": saturation of a HLS image
  - GCI_LightnessBand / "Lightness": lightness of a HLS image

- GCI_CyanBand / "Cyan": cyan portion of a CMY or CMYK image
- GCI_MagentaBand / "Magenta": magenta portion of a CMY or CMYK image
- GCI_YellowBand / "Yellow": yellow portion of a CMY or CMYK image, or yellow spectral band [0.58 - 0.62 um]
- GCI_BlackBand / "Black": black portion of a CMYK image.

Below values have been added in GDAL 3.10:

- GCI_PanBand / "Pan": Panchromatic band [0.40 - 1.00 um]
- GCI_CoastalBand / "Coastal": Coastal band [0.40 - 0.45 um]
- GCI_RedEdgeBand / "RedEdge": Red-edge band [0.69 - 0.79 um]
- GCI_NIRBand / "NIR": Near-InfraRed (NIR) band [0.75 - 1.40 um]
- GCI_SWIRBand / "SWIR": Short-Wavelength InfraRed (SWIR) band [1.40 - 3.00 um]
- GCI_MWIRBand / "MWIR": Mid-Wavelength InfraRed (MWIR) band [3.00 - 8.00 um]
- GCI_LWIRBand / "LWIR": Long-Wavelength InfraRed (LWIR) band [8.00 - 15 um]
- GCI_TIRBand / "TIR": Thermal InfraRed (TIR) band (MWIR or LWIR) [3 - 15 um]
- GCI_OtherIRBand / "OtherIR": Other infrared band [0.75 - 1000 um]
- GCI_SAR_Ka_Band / "SAR_Ka": Synthetic Aperture Radar (SAR) Ka band [0.8 - 1.1 cm / 27 - 40 GHz]
- GCI_SAR_K_Band / "SAR_K": Synthetic Aperture Radar (SAR) K band [1.1 - 1.7 cm / 18 - 27 GHz]
- GCI_SAR_Ku_Band / "SAR_Ku": Synthetic Aperture Radar (SAR) Ku band [1.7 - 2.4 cm / 12 - 18 GHz]
- GCI_SAR_X_Band / "SAR_X": Synthetic Aperture Radar (SAR) X band [2.4 - 3.8 cm / 8 - 12 GHz]
- GCI_SAR_C_Band / "SAR_C": Synthetic Aperture Radar (SAR) C band [3.8 - 7.5 cm / 4 - 8 GHz]
- GCI_SAR_S_Band / "SAR_S": Synthetic Aperture Radar (SAR) S band [7.5 - 15 cm / 2 - 4 GHz]
- GCI_SAR_L_Band / "SAR_L": Synthetic Aperture Radar (SAR) L band [15 - 30 cm / 1 - 2 GHz]
- GCI_SAR_P_Band / "SAR_P": Synthetic Aperture Radar (SAR) P band [30 - 100 cm / 0.3 - 1 GHz]

For spectral bands, the wavelength ranges are indicative only, and may vary depending on sensors. The `CENTRAL_WAVELENGTH_UM` and `FWHM_UM` metadata items in the band `IMAGERY` metadata domain of the raster band, when present, will give more accurate characteristics.

Values belonging to the IR domain are in the [GCI_IR_Start, GCI_IR_End] range. Values belonging to the SAR domain are in the [GCI_SAR_Start, GCI_SAR_End] range.

- A *color table*, if available.
- A *raster attribute table*, if available.
- *Reduced resolution overviews (pyramids)*, if available.

8.4.7 Color Table

A color table consists of zero or more color entries described in C by the following structure:

```
typedef struct
{
    /* gray, red, cyan or hue */
    short      c1;
```

(continues on next page)

(continued from previous page)

```
    /* green, magenta, or lightness */
    short      c2;

    /* blue, yellow, or saturation */
    short      c3;

    /* alpha or black band */
    short      c4;
} GDALColorEntry;
```

The color table also has a palette interpretation value (GDALPaletteInterp) which is one of the following values, and indicates how the c1/c2/c3/c4 values of a color entry should be interpreted.

- GPI_Gray: Use c1 as gray scale value.
- GPI_RGB: Use c1 as red, c2 as green, c3 as blue and c4 as alpha.
- GPI_CMYK: Use c1 as cyan, c2 as magenta, c3 as yellow and c4 as black.
- GPI_HLS: Use c1 as hue, c2 as lightness, and c3 as saturation.

To associate a color with a raster pixel, the pixel value is used as a subscript into the color table. That means that the colors are always applied starting at zero and ascending. There is no provision for indicating a pre-scaling mechanism before looking up in the color table.

8.4.8 Raster Attribute Table

A raster attribute table (or RAT) is used to encapsulate a table used to provide attribute information about pixel values. Each row in the table applies to a range of pixel values (or a single value in some cases), and might have attributes such as the histogram count for that range, the color pixels of that range should be drawn names of classes or any other generic information.

In C++, a RAT is accessible through the *GDALRasterAttributeTable* interface.

Raster attribute tables can be used to represent histograms, color tables, and classification information.

Each column in a raster attribute table has a name, a type (*GDALRATFieldType*) (integer, floating point, string, boolean, date time, geometries encoded as WKB), and a field usage (*GDALRATFieldUsage*). The usage distinguishes columns with particular understood purposes (such as color, histogram count, name) and columns that have specific purposes not understood by the library (long label, suitability_for_growing_wheat, etc).

In the general case each row has a column indicating the minimum pixel values falling into that category, and a column indicating the maximum pixel value. These are indicated with usage values of `GFU_Min`, and `GFU_Max`. In other cases where each row is a discrete pixel value, one column of usage `GFU_MinMax` can be used.

In other cases all the categories are of equal size and regularly spaced and the categorization information can be determined just by knowing the value at which the categories start, and the size of a category. This is called "Linear Binning" and the information is kept specially on the raster attribute table as a whole.

RATs are normally associated with *GDALRasterBand* and can be queried using the *GDALRasterBand::GetDefaultRAT()* method.

8.4.9 Overviews

A band may have zero or more overviews. Each overview is represented as a "free standing" *GDALRasterBand*. The size (in pixels and lines) of the overview will be different than the underlying raster, but the geographic region covered by overviews is the same as the full resolution band.

The overviews are used to display reduced resolution overviews more quickly than could be done by reading all the full resolution data and downsampling.

Bands also have a HasArbitraryOverviews property which is TRUE if the raster can be read at any resolution efficiently but with no distinct overview levels. This applies to some FFT encoded images, or images pulled through gateways where downsampling can be done efficiently at the remote point.

How overviews are selected depends on the software logic. The *GDALRasterBand::RasterIO()* or *GDALDataset::RasterIO()* methods document their logic.

8.5 GDAL Band Algebra

Added in version 3.12.

8.5.1 Description

Common algebraic operations can be on *GDALRasterBand* instances from C++ and Python:

- addition, with + operator
- subtraction, with * operator
- multiplication, with - operator
- division, with / operator
- absolute value / module, with *gdal::abs()* in C++ and *osgeo.gdal.abs()* in Python
- square root, with *gdal::sqrt()* in C++ and *osgeo.gdal.sqrt()* in Python
- logarithm base 10, with *gdal::log10()* in C++ and *osgeo.gdal.log10()* in Python
- natural logarithm, with *gdal::log()* in C++ and *osgeo.gdal.log()* in Python
- raising to the power, with *gdal::pow()* in C++ and *osgeo.gdal.pow()* in Python
- strictly greater than comparison, with > operator
- greater or equal to comparison, with >= operator
- strictly lesser than comparison, with < operator
- lesser of equal to comparison, with <= operator
- equal to comparison, with == operator
- not equal to comparison, with != operator
- logical and, with && operator in C++ and *osgeo.gdal.logical_and()* in Python
- logical or with || operator in C++ and *osgeo.gdal.logical_or()* in Python
- logical not, with ! operator in C++ and *osgeo.gdal.logical_not()* in Python

Left and right operands can be a Band object or a numeric value (at least one of them must be a Band object). If two bands are provided they must have the same dimension.

The result of those operations is a band of the same dimension as the input one(s), and whose each pixel is evaluated by applying the operator on the corresponding pixel in the input band(s). The resulting Band object is lazy evaluated, that is pixel values are computed only when they are requested.

Other operations are available:

- ternary / if-then-else, with *gdal::IfThenElse()* in C++ and *osgeo.gdal.where()* in Python (similar to NumPy where)

- cast to a data type, with *GDALRasterBand::AsType()* in C++ and *osgeo.gdal.Band.astype()* in Python
- minimum of several bands (or constants), with *gdal::min()* in C++ and *osgeo.gdal.minimum()* in Python
- maximum of several bands (or constants), with *gdal::max()* in C++ and *osgeo.gdal.maximum()* in Python
- arithmetic mean of several bands, with *gdal::mean()* in C++ and *osgeo.gdal.mean()* in Python

It is possible to serialize the operation to a *VRT -- GDAL Virtual Format* file by using *GDALDriver::CreateCopy()* on the dataset owing the result band.

When several bands are combined together, that at least one of them has a nodata value but they do not share the same nodata value, not-a-number will be used as the nodata value for the result band.

The capability is similar to the one offered by the *gdal raster calc* program.

> **Note**
>
> The comparison operators, including the ternary one, require a GDAL build against the muparser library.

> **Note**
>
> The operations are also available in the C API, for potential bindings to other languages. Cf *GDALRasterBandUnaryOp()*, *GDALRasterBandBinaryOpBand()*, *GDALRasterBandBinaryOpDouble()*, *GDALRasterBandBinaryOpDoubleToBand()*, *GDALRasterBandIfThenElse()*, *GDALRasterBandAsDataType()*, *GDALMaximumOfNBands()*, *GDALRasterBandMaxConstant()*, *GDALMinimumOfNBands()*, *GDALRasterBandMinConstant()* and *GDALMeanOfNBands()*

8.5.2 Examples

8.5.2.1 Example 1: Convert a RGB dataset to a graylevel one.

C++

Python

```
#include <gdal_priv.h>

int main()
{
    GDALAllRegister();

    auto poDS = std::unique_ptr<GDALDataset>(GDALDataset::Open("rgb.tif"));
    auto& R = *(poDS->GetRasterBand(1));
    auto& G = *(poDS->GetRasterBand(2));
    auto& B = *(poDS->GetRasterBand(3));
    auto graylevel = (0.299 * R + 0.587 * G + 0.114 * B).AsType(GDT_UInt8);

    auto poGTiffDrv = GetGDALDriverManager()->GetDriverByName("GTiff");
    std::unique_ptr<GDALDataset>(
        poGTiffDrv->CreateCopy("graylevel.tif", graylevel.GetDataset(), false, nullptr,␣
↪nullptr, nullptr)).reset();

    return 0;
}
```

```
from osgeo import gdal
gdal.UseExceptions()

with gdal.Open("rgb.tif") as ds:
   R = ds.GetRasterBand(1)
   G = ds.GetRasterBand(2)
   B = ds.GetRasterBand(3)
   graylevel = (0.299 * R + 0.587 * G + 0.114 * B).astype(gdal.GDT_UInt8)
   gdal.GetDriverByName("GTiff").CreateCopy("graylevel.tif", graylevel)
```

8.5.2.2 Example 2: Compute normalized difference vegetation index (NDVI)

C++

Python

```
#include <gdal_priv.h>

int main()
{
    GDALAllRegister();

    auto poDS = std::unique_ptr<GDALDataset>(GDALDataset::Open("rgbnir.tif"));
    auto& R = *(poDS->GetRasterBand(1));
    auto& NIR = *(poDS->GetRasterBand(4));
    auto NDVI = (NIR - R) / (NIR + R);

    auto poGTiffDrv = GetGDALDriverManager()->GetDriverByName("GTiff");
    std::unique_ptr<GDALDataset>(
        poGTiffDrv->CreateCopy("NDVI.tif", NDVI.GetDataset(), false, nullptr, nullptr,␣
↪nullptr)).reset();

    return 0;
}
```

```
from osgeo import gdal
gdal.UseExceptions()

with gdal.Open("rgbnir.tif") as ds:
   R = ds.GetRasterBand(1)
   NIR = ds.GetRasterBand(4)
   NDVI = (NIR - R) / (NIR + R)
   gdal.GetDriverByName("GTiff").CreateCopy("NDVI.tif", NDVI)
```

8.5.2.3 Example 3: Normalizing the values of a band to the [0, 1] range using the minimum and maximum of all bands

C++

Python

```
#include <gdal_priv.h>
```

(continues on next page)

(continued from previous page)

```
int main()
{
    GDALAllRegister();

    auto poDS = std::unique_ptr<GDALDataset>(GDALDataset::Open("input.tif"));
    auto& A = *(poDS->GetRasterBand(1));
    auto& B = *(poDS->GetRasterBand(2));
    auto& C = *(poDS->GetRasterBand(3));
    auto max_minus_min = gdal::max(A,B,C) - gdal::min(A,B,C);
    auto A_normalized = gdal::IfThenElse(max_minus_min == 0, 1.0, (A - gdal::min(A,B,C))␣
↪/ max_minus_min);

    auto poVRTDrv = GetGDALDriverManager()->GetDriverByName("VRT");
    std::unique_ptr<GDALDataset>(
        poVRTDrv->CreateCopy("A_normalized.vrt", A_normalized.GetDataset(), false,␣
↪nullptr, nullptr, nullptr)).reset();

    return 0;
}
```

```
from osgeo import gdal
gdal.UseExceptions()

with gdal.Open("input.tif") as ds:
   A = ds.GetRasterBand(1)
   B = ds.GetRasterBand(2)
   C = ds.GetRasterBand(3)
   max_minus_min = gdal.maximum(A,B,C) - gdal.minimum(A,B,C)
   A_normalized = gdal.where(max_minus_min == 0, 1.0, (A - gdal.min(A,B,C)) / max_minus_
↪min)
   gdal.GetDriverByName("VRT").CreateCopy("A_normalized.vrt", A_normalized)
```

8.6 Multidimensional Raster Data Model

This document attempts to describe the GDAL multidimensional data model, that has been added in GDAL 3.1. That is the types of information that a GDAL multidimensional dataset can contain, and their semantics.

The multidimensional raster API is a generalization of the traditional *Raster Data Model*, to address 3D, 4D or higher dimension datasets. Currently, it is limited to basic read/write API, and is not that much plugged into other higher level utilities.

It is strongly inspired from the netCDF and HDF5 API and data models. See HDF5 format and data model.

A *GDALDataset* with multidimensional content contains a root *GDALGroup*.

8.6.1 Group

A *GDALGroup* (modelling a HDF5 Group) is a named container of GDALAttribute, GDALMDArray or other GDALGroup. Hence GDALGroup can describe a hierarchy of objects.

8.6.2 Attribute

A *GDALAttribute* (modelling a HDF5 Attribute) has a name and a value, and is typically used to describe a metadata item. The value can be (for the HDF5 format) in the general case a multidimensional array of "any" type (in most cases, this will be a single value of string or numeric type)

8.6.3 Multidimensional array

A *GDALMDArray* (modelling a HDF5 Dataset) has a name, a multidimensional array, references a number of GDALDimension, and has a list of GDALAttribute.

Most drivers use the row-major convention for dimensions: that is, when considering that the array elements are stored consecutively in memory, the first dimension is the slowest varying one (in a 2D image, the row), and the last dimension the fastest varying one (in a 2D image, the column). That convention is the default convention used for NumPy arrays, the MEM driver and the HDF5 and netCDF APIs. The GDAL API is mostly agnostic about that convention, except when passing a NULL array as the *stride* parameter for the *GDALAbstractMDArray::Read()* and *GDALAbstractMDArray::Write()* methods. You can refer to NumPy documentation about multidimensional array indexing order issues

A GDALMDArray has also optional properties:

- Coordinate reference system: *OGRSpatialReference*
- No data value:
- Unit
- Offset, such that unscaled_value = offset + scale * raw_value
- Scale, such that unscaled_value = offset + scale * raw_value

A GDALMDArray may also have zero or more overviews . Each overview is represented as a *GDALMDArray*, that may be "free standing" or attached to a group. Overview arrays have matching dimensions with the full resolution array, but with smaller size. The domain covered by overviews is the same as the full resolution array.

Number of operations can be applied on an array to get modified views of it: *GDALMDArray::Transpose()*, *GDALMDArray::GetView()*, etc.

The *GDALMDArray::Cache()* method can be used to cache the value of a view array into a sidecar file.

8.6.4 Dimension

A *GDALDimension* describes a dimension / axis used to index multidimensional arrays. It has the following properties:

- a name
- a size, that is the number of values that can be indexed along the dimension
- a type, which is a string giving the nature of the dimension. Predefined values are: HORIZONTAL_X, HORIZONTAL_Y, VERTICAL, TEMPORAL, PARAMETRIC Other values might be used. Empty value means unknown.
- a direction. Predefined values are: EAST, WEST, SOUTH, NORTH, UP, DOWN, FUTURE, PAST Other values might be used. Empty value means unknown.
- a reference to a GDALMDArray variable, typically one-dimensional, describing the values taken by the dimension. For a georeferenced GDALMDArray and its X dimension, this will be typically the values of the easting/longitude for each grid point.

8.6.5 Data Type

A *GDALExtendedDataType* (modelling a HDF5 datatype) describes the type taken by an individual value of a GDALAttribute or GDALMDArray. Its class can be NUMERIC, STRING or COMPOUND. For NUMERIC, the existing *GDALDataType* enumerated values are supported. For COMPOUND, the data type is a list of members, each member being described by a name, a offset in byte in the compound structure and a GDALExtendedDataType.

> **Note**
>
> The HDF5 modelisation allows for more complex datatypes.

> **Note**
>
> HDF5 does not have native data types for complex values whereas GDALDataType does. So a driver may decide to expose a GDT_Cxxxx datatype from a HDF5 Compound data type representing a complex value.

8.6.6 Objects lifetime

Driver implementations of the multidimensional API are such that instances of GDALGroup, GDALMDArray, GDALAttribute and GDALDimension can be used independently of the objects from which they have been obtained from. So:

- the GDALGroup instance returned by *GDALDataset::GetRootGroup()* can be used after the dataset has been closed.
- Sub-groups obtained from *GDALGroup::OpenGroup()* or arrays obtained from *GDALGroup::OpenMDArray()* can be used after the owning group has been released.
- Similarly for attributes or dimensions obtained from a group or an array.

So the following is perfectly legal:

```
#include "gdal_priv.h"

std::shared_ptr<GDALMDArray> GetArray()
{
    auto poDataset = std::unique_ptr<GDALDataset>(
        GDALDataset::Open( "in.nc", GDAL_OF_MULTIDIM_RASTER ));
    if( !poDataset )
    {
        return nullptr:
    }
    auto poRootGroup = poDataset->GetRootGroup();
    if( !poRootGroup )
    {
        return nullptr:
    }
    auto poVar = poRootGroup->OpenMDArray("temperature");
    if( !poVar )
    {
        return nullptr:
    }
    return poVar;
```

(continues on next page)

(continued from previous page)

```
}

int main()
{
    GDALAllRegister();
    auto poVar = GetArray();
    if( !poVar )
    {
        exit(1);
    }

    // Do something with poVar

    return 0;
}
```

A potential point of attention is that, when creating / editing a dataset, all those objects keep alive the underlying file descriptors, so changes are only guaranteed to be serialized when all objects related to a dataset have been released.

C++

Python

```
int main()
{
    GDALAllRegister();

    auto poDataset = std::unique_ptr<GDALDataset>(
        GetGDALDriverManager()->GetDriverByName("netCDF")->
            CreateMultiDimensional( "new.nc", nullptr, nullptr ));
    auto poRootGroup = poDataset->GetRootGroup();

    // Could be closed a bit later too
    poDataset.reset();

    auto poDim = poRootGroup->CreateDimension(
        "my_dim", std::string(), std::string(), 10);
    auto poArray = poRootGroup->CreateMDArray(
        "my_var", {poDim}, GDALExtendedDataType::Create(GDT_UInt8), nullptr);

    // Can be closed in any order
    poArray.reset();
    poDim.reset();
    poRootGroup.reset();

    // new.nc is fully valid just now

    return 0;
}
```

```
from osgeo import gdal

with gdal.GetDriverByName("netCDF").CreateMultiDimensional("new.nc") as ds:
```

(continues on next page)

(continued from previous page)

```
    rg = ds.GetRootGroup()
    dim = rg.CreateDimension("my_dim", "", "", 10)
    array = rg.CreateMDArray("my_var", [dim], gdal.ExtendedDataType.Create(gdal.GDT_
↪UInt8))

    del array
    del dim
    del rg

# new.nc is fully valid just now
```

8.6.7 Differences with the GDAL 2D raster data model

- The concept of GDALRasterBand is no longer used for multidimensional. This can be modelled as either different GDALMDArray, or using a compound data type.
- There is not yet a standardized concept multi-resolution arrays/overviews in the multidimensional API

8.6.8 Bridges between GDAL 2D classic raster data model and multidimensional data model

The *`GDALRasterBand::AsMDArray()`* and *`GDALMDArray::AsClassicDataset()`* can be used to respectively convert a raster band to a MD array or a 2D dataset to a MD array.

8.6.9 Applications

The following applications can be used to inspect and manipulate multidimensional datasets:

- *gdalmdiminfo*
- *gdalmdimtranslate*

8.7 Vector Data Model

This page documents the classes used to handle vector data. Many data types and method names are based on the OGC Simple Features data model, so it may be helpful to review the specifications published by OGC. For historical reasons, GDAL uses the "OGR" prefix to denote types and functions that apply only to vector data.

8.7.1 Class Overview

The following classes form the core of the vector data model:

- Geometry (*ogr_geometry.h*): The geometry classes (*`OGRGeometry`*, etc) encapsulate the OGC vector data types. They provide some geometry operations and translation to/from well known binary and text format. A geometry includes a spatial reference system (projection).
- Spatial Reference (*ogr_spatialref.h*): An *`OGRSpatialReference`* encapsulates the definition of a projection and datum.
- Feature (*ogr_feature.h*): The *`OGRFeature`* encapsulates the definition of a whole feature, that is a set of geometries and attributes relating to a single entity.
- Feature Class Definition (*ogr_feature.h*): The *`OGRFeatureDefn`* class captures the schema (set of field definitions) for a group of related features (normally a whole layer).
- Layer (*ogrsf_frmts.h*): *`OGRLayer`* is an abstract class representing a layer of features in a *`GDALDataset`*.

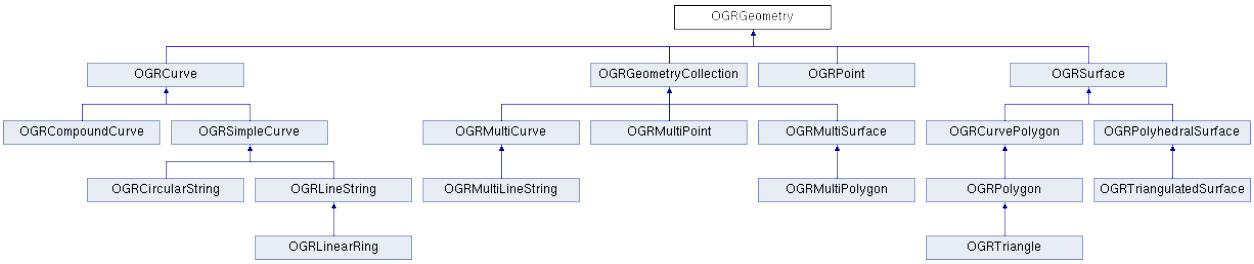
- Dataset (*gdal_priv.h*): A `GDALDataset` is an abstract base class representing a file or database containing one or more `OGRLayer` objects.
- Drivers (*gdal_priv.h*): A `GDALDriver` represents a translator for a specific format, capable of opening and possibly writing `GDALDataset` objects. All available drivers are managed by the `GDALDriverManager`.

8.7.2 Geometry

Individual geometry classes are used to represent the different types of vector geometry. All the geometry classes derive from `OGRGeometry` which defines the common functionality of all geometries. Geometry types include `OGRPoint`, `OGRLineString`, `OGRPolygon`, `OGRGeometryCollection`, `OGRMultiPoint`, `OGRMultiLineString`, `OGRMultiPolygon`, and `OGRPolyhedralSurface`. The special case of a triangular polygon can be represented as a `OGRTriangle`, a non-overlapping collection of which can be represented by an `OGRTriangulatedSurface`. An additional set of types is used to store non-linear geometries: `OGRCircularString`, `OGRCompoundCurve`, `OGRCurvePolygon`, `OGRMultiCurve` and `OGRMultiSurface`.

Any of the above geometry classes can store coordinates in two (XY), three (XYZ or XYM), or four (XYZM) dimensions.

Additional intermediate classes contain functionality that is used by multiple geometry types. These include `OGRCurve` (base class for `OGRLineString`) and `OGRSurface` (base class for `OGRPolygon`). Some intermediate interfaces modeled in the simple features abstract model and SFCOM are not modeled in OGR at this time. In most cases the methods are aggregated into other classes.



The `OGRGeometryFactory` is used to convert well known text (WKT) and well known binary (WKB) format data into the appropriate `OGRGeometry` subclass. These are predefined ASCII and binary formats for representing all the types of simple features geometries.

The `OGRGeometry` includes a reference to an `OGRSpatialReference` object, defining the spatial reference system of that geometry. This is normally a reference to a shared spatial reference object with reference counting for each of the `OGRGeometry` objects using it. Note however that in the general case, all geometric processing done by GDAL is done in a planar way, ignoring potential discontinuity issues at the poles or the antimeridian. `OGRGeometryFactory::transformWithOptions()` can be used in some cases to split geometries at the poles or the antimeridian.

While it is theoretically possible to derive other or more specific geometry classes from the existing `OGRGeometry` classes, this isn't an aspect that has been well thought out. In particular, it would be possible to create specialized classes using the `OGRGeometryFactory` without modifying it.

8.7.2.1 Compatibility issues with non-linear geometries

Generic mechanisms have been introduced so that creating or modifying a feature with a non-linear geometry in a layer of a driver that does not support it will transform that geometry in the closest matching linear geometry. This linearization can be controlled using *Vector related options*.

On the other side, when retrieving data from the OGR C API, the `OGRSetNonLinearGeometriesEnabledFlag()` function can be used, so that geometries and layer geometry type returned are also converted to their linear approximation if necessary.

8.7.3 Spatial Reference

The *OGRSpatialReference* class is intended to store an OpenGIS Spatial Reference System definition. Currently local, geographic and projected coordinate systems are supported. Vertical coordinate systems, geocentric coordinate systems, and compound (horizontal + vertical) coordinate systems are as well supported in recent GDAL versions.

The spatial coordinate system data model is inherited from the OpenGIS Well Known Text format. A simple form of this is defined in the Simple Features specifications. A more sophisticated form is found in the Coordinate Transformation specification. The *OGRSpatialReference* is built on the features of the Coordinate Transformation specification but is intended to be compatible with the earlier simple features form.

There is also an associated *OGRCoordinateTransformation* class that encapsulates use of PROJ for converting between different coordinate systems.

8.7.4 Feature / Feature Definition

The *OGRGeometry* captures the geometry of a vector feature. The *OGRFeature* contains geometry, and adds feature attributes, feature id, and a feature class identifier. It may also contain styling information. Several geometries can be associated with an *OGRFeature*.

The set of attributes (*OGRFieldDefn*), their types, names and so forth is represented via the *OGRFeatureDefn* class. One *OGRFeatureDefn* normally exists for a layer of features. The same definition is shared in a reference counted manner by the feature of that type (or feature class).

The feature id (FID) of a feature is intended to be a unique identifier for the feature within the layer it is a member of. Freestanding features, or features not yet written to a layer may have a null (OGRNullFID) feature id. The feature ids are modeled in OGR as a 64-bit integer; however, this is not sufficiently expressive to model the natural feature ids in some formats. For instance, the GML feature id is a string.

The *OGRFeatureDefn* also contains an indicator of the types of geometry allowed for that feature class (returned as an *OGRwkbGeometryType* from *OGRFeatureDefn::GetGeomType()*). If this is *OGRwkbGeometryType::wkbUnknown* then any type of geometry is allowed. This implies that features in a given layer can potentially be of different geometry types though they will always share a common attribute schema.

Several geometry fields (*OGRGeomFieldDefn*) can be associated with an *OGRFeatureDefn*. Each geometry field has its own indicator of geometry type allowed, returned by *OGRGeomFieldDefn::GetType()*, and its spatial reference system, returned by *OGRGeomFieldDefn::GetSpatialRef()*.

The *OGRFeatureDefn* also contains a feature class name (normally used as a layer name).

8.7.5 Field Definitions

The behavior of each field in a feature class is defined by a shared *OGRFieldDefn*. The *OGRFieldDefn* specifies the field type from the values of *OGRFieldType*. Values stored in this field may be further restricted according to a *OGRFieldSubType*. For example, a field may have a type of *OGRFieldType::OFTInteger* with a subtype of *OGRFieldSubType::OFSTBoolean*.

The *OGRFieldDefn* can also track whether a field is allowed to be null (*OGRFieldDefn::IsNullable()*), whether its value must be unique (*OGRFieldDefn::IsUnique()*), and formatting information such as the number of decimal digits, width, and justification. It may also define a default value in case one is not manually specified.

8.7.5.1 Field Domains

Some formats support the use of field domains that describe the values that can be stored in a given attribute field. An *OGRFieldDefn* may reference a single *OGRFieldDomain* that is associated with a *GDALDataset*. Programs using GDAL may use the *OGRFieldDomain* to appropriately constrain user input. GDAL does not perform validation itself and will allow the storage of values that violate a field's associated *OGRFieldDomain*.

Available types of *OGRFieldDomain* include:

- *OGRCodedFieldDomain*, which constrains values those present in a specified enumeration
- *OGRRangeFieldDomain*, which constrains values to a specified range
- *OGRGlobFieldDomain*, which constrains values to those matching a specified pattern

Additionally, an *OGRFieldDomain* may define policies describing the values that should be assigned to domain-controlled fields when features are split or merged.

8.7.6 Layer

An *OGRLayer* represents a layer of features within a data source. All features in an *OGRLayer* share a common schema and are of the same *OGRFeatureDefn*. An *OGRLayer* class also contains methods for reading features from the data source. The *OGRLayer* can be thought of as a gateway for reading and writing features from an underlying data source such as a file on disk, or the result of a database query.

The *OGRLayer* includes methods for sequential and random reading and writing. Read access (via the *OGRLayer::GetNextFeature()* method) normally reads all features, one at a time sequentially; however, it can be limited to return features intersecting a particular geographic region by installing a spatial filter on the *OGRLayer* (via the *OGRLayer::SetSpatialFilter()* method). A filter on attributes can only be set with the *OGRLayer::SetAttributeFilter()* method. By default, all available attributes and geometries are read but this can be controlled by flagging fields as ignored (*OGRLayer::SetIgnoredFields()*).

Starting with GDAL 3.6, as an alternative to getting features through `GetNextFeature`, it is possible to retrieve them by batches, with a column-oriented memory layout, using the *OGRLayer::GetArrowStream()* method (cf *Reading From OGR using the Arrow C Stream data interface*).

An *OGRLayer* may also store an *OGRStyleTable* that provides a set of styles that may be used by features in the layer. More information on GDAL's handling of feature styles can be found in the *Feature Style Specification*.

One flaw in the current OGR architecture is that the spatial and attribute filters are set directly on the *OGRLayer* which is intended to be the only representative of a given layer in a data source. This means it isn't possible to have multiple read operations active at one time with different spatial filters on each.

Another question that might arise is why the *OGRLayer* and *OGRFeatureDefn* classes are distinct. An *OGRLayer* always has a one-to-one relationship to an *OGRFeatureDefn*, so why not amalgamate the classes? There are two reasons:

- As defined now *OGRFeature* and *OGRFeatureDefn* don't depend on *OGRLayer*, so they can exist independently in memory without regard to a particular layer in a data store.
- The SF CORBA model does not have a concept of a layer with a single fixed schema the way that the SFCOM and SFSQL models do. The fact that features belong to a feature collection that is potentially not directly related to their current feature grouping may be important to implementing SFCORBA support using OGR.

The *OGRLayer* class is an abstract base class. An implementation is expected to be subclassed for each file format driver implemented. OGRLayers are normally owned directly by their *GDALDataset*, and aren't instantiated or destroyed directly.

8.7.7 Dataset

A *GDALDataset* represents a set of *OGRLayer* objects. This usually represents a single file, set of files, database or gateway. A *GDALDataset* has a list of *OGRLayer* which it owns but can return references to.

GDALDataset is an abstract base class. An implementation is expected to be subclassed for each file format driver implemented. *GDALDataset* objects are not normally instantiated directly but rather with the assistance of an *GDALDriver*. Deleting an *GDALDataset* closes access to the underlying persistent data source, but does not normally result in deletion of that file.

A *GDALDataset* has a name (usually a filename or database connection string) that can be used to reopen the data source with a *GDALDriver*.

The *GDALDataset* also has support for executing a datasource specific command, normally a form of SQL. This is accomplished via the *GDALDataset::ExecuteSQL()* method. While some datasources (such as PostGIS and Oracle) pass the SQL through to an underlying database, OGR also includes support for evaluating a subset of the SQL SELECT statement against any datasource (see *OGR SQL dialect and SQLITE SQL dialect*.)

When using some drivers, the *GDALDataset* also offers a mechanism for to start, commit, and rollback transactions when interacting with the underlying data store.

A *GDALDataset* may also be aware of relationships between layers (e.g., a foreign key relationship between database tables). Information about these relationships is stored in a *GDALRelationship*.

> **Note**
>
> Earlier versions of GDAL represented vector datasets using the `OGRDataSource` class. This class has been maintained for backwards compatibility but is functionally equivalent to a *GDALDataset* for vector data.

8.7.8 Drivers

A *GDALDriver* object is instantiated for each file format supported. The *GDALDriver* objects are registered with the *GDALDriverManager*, a singleton class that is normally used to open new datasets.

It is intended that a new *GDALDriver* object is instantiated and define function pointers for operations like Identify(), Open() for each file format to be supported (along with a file format specific *GDALDataset*, and *OGRLayer* classes).

On application startup registration functions are normally called for each desired file format. These functions instantiate the appropriate *GDALDriver* objects, and register them with the *GDALDriverManager*. When a dataset is to be opened, the driver manager will normally try each *GDALDataset* in turn, until one succeeds, returning a *GDALDataset* object.

8.8 Geometry Validity

Several functions and utilities in GDAL deal with the concepts of "valid" and "invalid" geometries. Validity in this context refers to the OGR Simple Features standard. Incorrect results may be obtained when invalid geometries are used in spatial algorithms such as intersection testing/computation.

Geometry validity can be checked using the *gdal vector check-geometry* command-line utility or the API functions *OGRGeometry::IsValid()* (C++) and `ogr.Geometry.IsValid()` (Python). In many cases, invalid geometries can be repaired using the *gdal vector make-valid* command-line utility or the API functions *OGRGeometry::MakeValid()* (C++) and `ogr.Geometry.MakeValid()` (Python).

Geometry validity does not consider interactions between features, such as gaps and overlaps, although *gdal vector check-coverage* provides some means for evaluating this. Qualitative errors such as spikes and slivers are outside the scope of validity testing.

8.8.1 Validity checking

GDAL relies on the GEOS library to check the validity of geometries. This library is widely used by open source geospatial software, so geometries considered valid by GDAL should be considered valid by QGIS, PostGIS, shapely, etc. Still, some geometries considered valid by GEOS may not be considered valid in other software. The following limitations apply to validity testing:

- GEOS does not enforce any particular ring orientation for polygon shells and holes, and permits repeated points within a ring.
- GEOS considers only two dimensions when checking geometry validity. For example, if two elements of a MultiPolygon occupy the same space but with different Z values, they will still be considered invalid by GEOS and therefore GDAL.

- Curved geometries are approximated as linear geometries before being evaluated by GEOS. Linearized geometries may be valid where the original geometries are not, and vice-versa.

Finally, not all software uses the OGC model for invalid geometries. As a simple example, the self-touching ring shown *below* is considered valid in the Esri geometry model.

8.8.2 Geometry repair

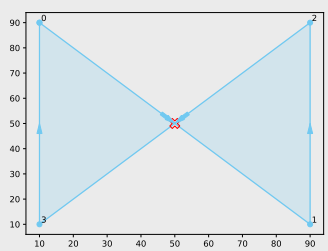
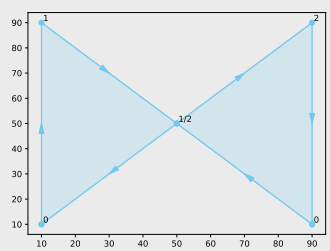
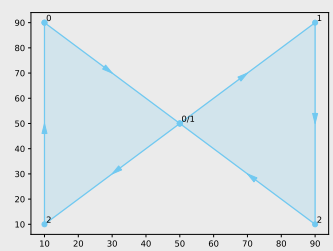
In many cases, the correct representation of an invalid geometry is ambiguous. GEOS provides two different algorithms for constructing a valid geometry from an invalid one. The "linework" method attempts to preserve as many of the input lines as possible, in some cases converting polygon holes into shells and vice-versa. On the other hand, the "structure" method can remove input lines such that areas covered by a polygon shell in the input remain covered in the output.

In some cases, such as a polygon with a non-closed ring, an invalid geometry cannot be represented in the GEOS library and therefore cannot be repaired automatically.

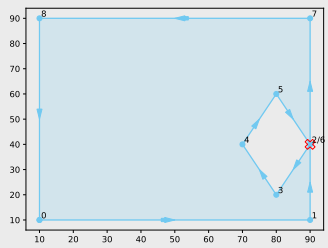
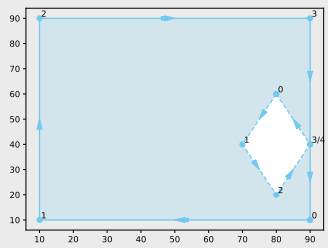
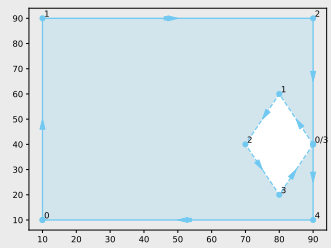
8.8.2.1 Invalid geometry examples

The cases below illustrate a number of topological invalidities, how they are reported by *gdal vector check-geometry*, and how they may be resolved by the different methods of *gdal vector make-valid*.

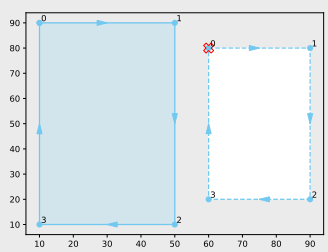
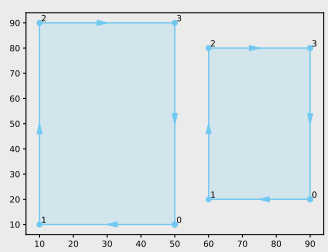
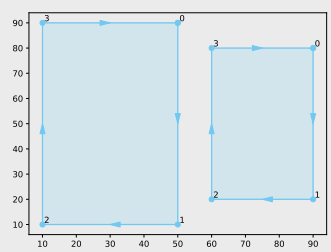
Self-intersecting polygon

Input geometry and result of `gdal vector check-geometry`	`gdal vector make-valid --method=linework`	`gdal vector make-valid --method=structure`
		
Input geometry: `POLYGON ((10 90,90 10,90 90,10 10,10 90))` Error message: `Self-intersection` Error geometry: `MULTIPOINT (50 50)`	`MULTIPOLYGON (((10 10,10 90,50 50,10 10)),((90 10,50 50,90 90,90 10)))`	`MULTIPOLYGON (((10 90,50 50,10 10,10 90)),((50 50,90 90,90 10,50 50)))`

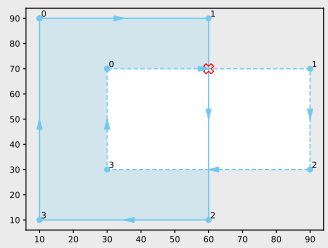
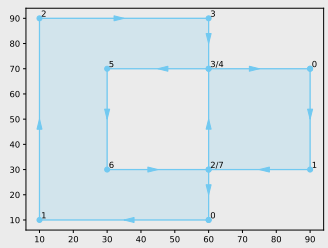
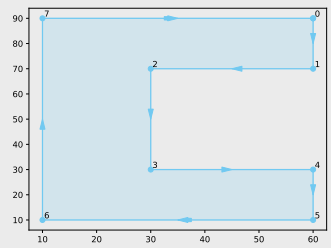
Polygon with self-touching ring

Input geometry and result of `gdal vector check-geometry`	`gdal vector make-valid --method=linework`	`gdal vector make-valid --method=structure`
		
Input geometry: `POLYGON ((10 10,90 10,90 40,80 20,70 40, 80 60,90 40,90 90,10 90,10 10))` Error message: `Ring Self-intersection` Error geometry: `MULTIPOINT (90 40)`	`POLYGON ((90 10,10 10,10 90,90 90,90 40,90 10),(80 60,70 40,80 20,90 40,80 60))`	`POLYGON ((10 10,10 90,90 90,90 40,90 10,10 10),(90 40,80 60,70 40,80 20,90 40))`

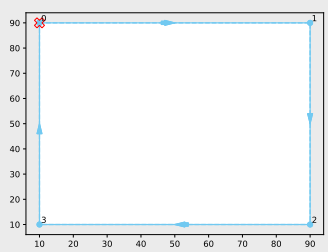
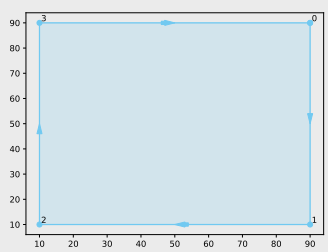
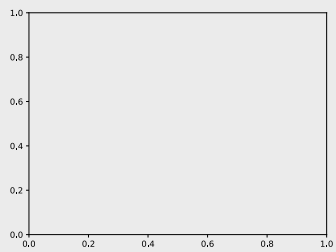
Polygon hole outside shell

Input geometry and result of `gdal vector check-geometry`	`gdal vector make-valid --method=linework`	`gdal vector make-valid --method=structure`
		
Input geometry: `POLYGON ((10 90,50 90,50 10,10 10,10 90),(60 80,90 80,90 20,60 20,60 80))` Error message: `Hole lies outside shell` Error geometry: `MULTIPOINT (60 80)`	`MULTIPOLYGON (((50 10,10 10,10 90,50 90,50 10)),((90 20,60 20,60 80,90 80,90 20)))`	`MULTIPOLYGON (((50 90,50 10,10 10,10 90,50 90)),((90 80,90 20,60 20,60 80,90 80)))`

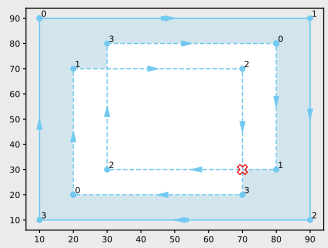
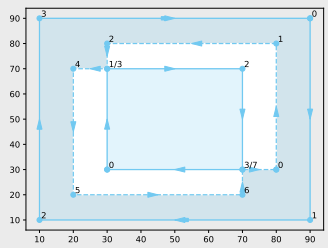
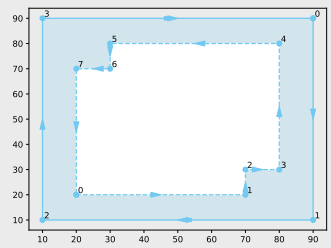
Hole partially outside polygon shell

Input geometry and result of `gdal vector check-geometry`	`gdal vector make-valid --method=linework`	`gdal vector make-valid --method=structure`
		
Input geometry: `POLYGON ((10 90,60 90,60 10,10 10,10 90),(30 70,90 70,90 30,30 30,30 70))` Error message: `Self-intersection` Error geometry: `MULTIPOINT (60 70)`	`MULTIPOLYGON (((90 70,90 30,60 30,60 70,90 70)),((60 10,10 10,10 90,60 90,60 70,30 70,30 30,60 30,60 10)))`	`POLYGON ((60 90,60 70,30 70,30 30,60 30,60 10,10 10,10 90,60 90))`

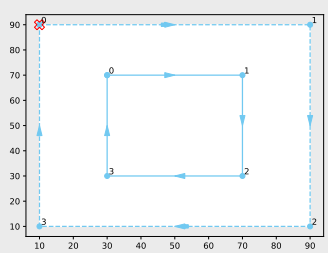
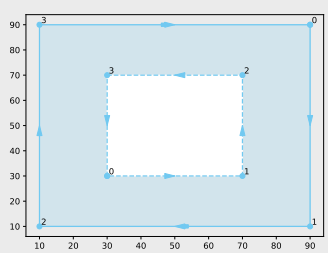
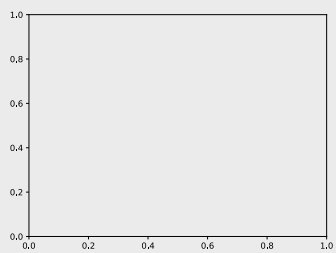
Polygon hole equal to shell

Input geometry and result of `gdal vector check-geometry`	`gdal vector make-valid --method=linework`	`gdal vector make-valid --method=structure`
		
Input geometry: `POLYGON ((10 90,90 90,90 10,10 10,10 90),(10 90,90 90,90 10,10 10,10 90))` Error message: `Self-intersection` Error geometry: `MULTIPOINT (10 90)`	`POLYGON ((90 90,90 10,10 10,10 90,90 90))`	`POLYGON EMPTY`

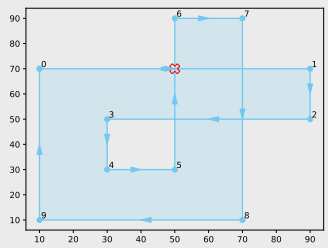
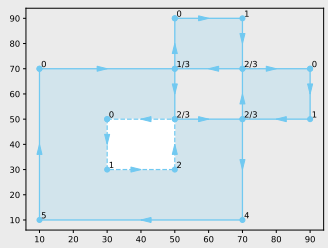
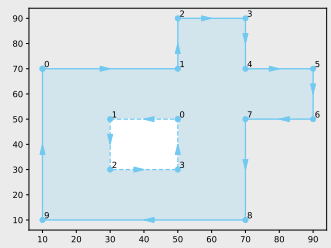
Polygon holes overlap

Input geometry and result of `gdal vector check-geometry`	`gdal vector make-valid --method=linework`	`gdal vector make-valid --method=structure`
		
Input geometry: `POLYGON ((10 90,90 90,90 10,10 10,10 90),(80 80,80 30,30 30,30 80,80 80),(20 20,20 70,70 70,70 20,20 20))` Error message: `Self-intersection` Error geometry: `MULTIPOINT (70 30)`	`MULTIPOLYGON (((90 90,90 10,10 10,10 90,90 90),(80 30,80 80,30 80,30 70,20 70,20 20,70 20,70 30,80 30)),((30 30,30 70,70 70,70 30,30 30)))`	`POLYGON ((90 90,90 10,10 10,10 90,90 90),(20 20,70 20,70 30,80 30,80 80,30 80,30 70,20 70,20 20))`

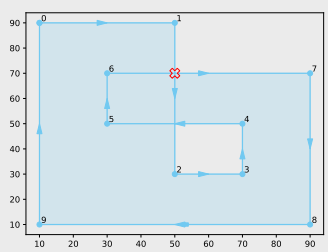
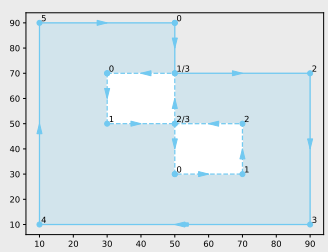
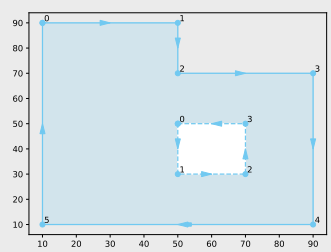
Polygon shell inside hole

Input geometry and result of `gdal vector check-geometry`	`gdal vector make-valid --method=linework`	`gdal vector make-valid --method=structure`
		
Input geometry: `POLYGON ((30 70,70 70,70 30,30 30,30 70),(10 90,90 90,90 10,10 10,10 90))` Error message: `Hole lies outside shell` Error geometry: `MULTIPOINT (10 90)`	`POLYGON ((90 90,90 10,10 10,10 90,90 90),(30 30,70 30,70 70,30 70,30 30))`	`POLYGON EMPTY`

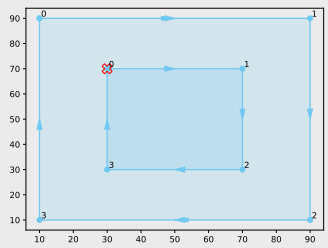
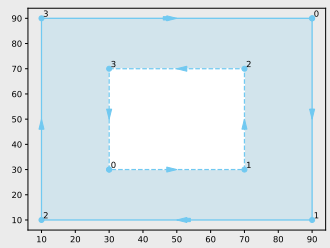
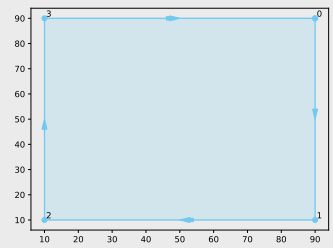
Self-crossing polygon shell

Input geometry and result of `gdal vector check-geometry`	`gdal vector make-valid --method=linework`	`gdal vector make-valid --method=structure`
		
Input geometry: `POLYGON ((10 70,90 70,90 50,30 50,30 30,50 30,50 90,70 90,70 10,10 10,10 70))` Error message: `Self-intersection` Error geometry: `MULTIPOINT (50 70)`	`MULTIPOLYGON (((10 70,50 70,50 50,70 50,70 10,10 10,10 70),(30 50,30 30,50 30,50 50,30 50)),((50 90,70 90,70 70,50 70,50 90)),((90 70,90 50,70 50,70 70,90 70)))`	`POLYGON ((10 70,50 70,50 90,70 90,70 70,90 70,90 50,70 50,70 10,10 10,10 70),(50 50,30 50,30 30,50 30,50 50))`

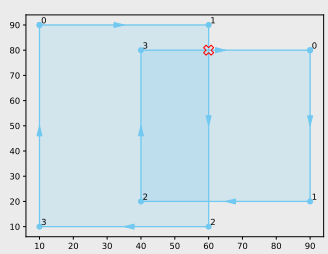
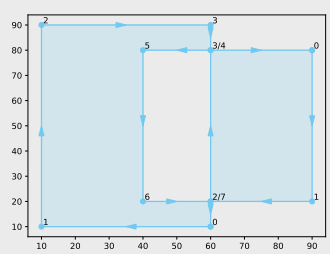
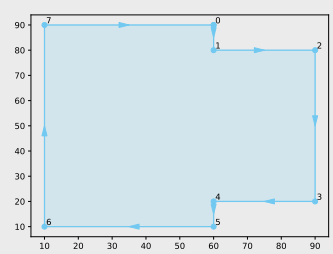
Self-overlapping polygon shell

Input geometry and result of `gdal vector check-geometry`	`gdal vector make-valid --method=linework`	`gdal vector make-valid --method=structure`
		
Input geometry: `POLYGON ((10 90,50 90,50 30,70 30,70 50,30 50,30 70,90 70,90 10,10 10,10 90))` Error message: `Self-intersection` Error geometry: `MULTIPOINT (50 70)`	`POLYGON ((50 90,50 70,90 70,90 10,10 10,10 90,50 90),(30 70,30 50,50 50,50 70,30 70),(50 30,70 30,70 50,50 50,50 30))`	`POLYGON ((10 90,50 90,50 70,90 70,90 10,10 10,10 90),(50 50,50 30,70 30,70 50,50 50))`

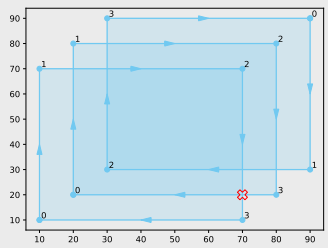
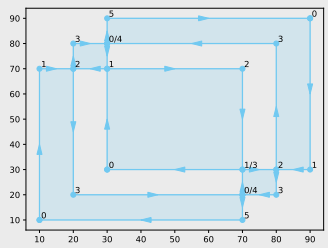
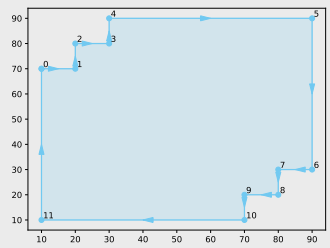
Nested MultiPolygons

Input geometry and result of `gdal vector check-geometry`	`gdal vector make-valid --method=linework`	`gdal vector make-valid --method=structure`
		
Input geometry: `MULTIPOLYGON (((30 70,70 70,70 30,30 30, 30 70)),((10 90,90 90,90 10,10 10,10 90)))` Error message: `Nested shells` Error geometry: `MULTIPOINT (30 70)`	`POLYGON ((90 90,90 10,10 10,10 90,90 90),(30 30,70 30,70 70,30 70,30 30))`	`MULTIPOLYGON (((90 90,90 10,10 10,10 90,90 90)))`

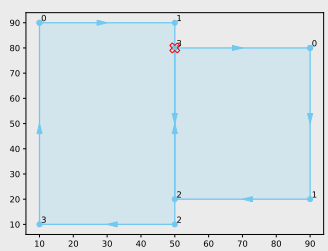
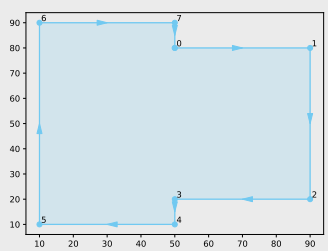
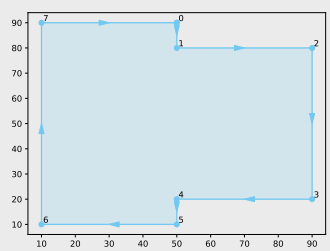
Overlapping MultiPolygons

Input geometry and result of `gdal vector check-geometry`	`gdal vector make-valid --method=linework`	`gdal vector make-valid --method=structure`
		
Input geometry: `MULTIPOLYGON (((10 90,60 90,60 10,10 10, 10 90)),((90 80,90 20,40 20,40 80,90 80)))` Error message: `Self-intersection` Error geometry: `MULTIPOINT (60 80)`	`MULTIPOLYGON (((90 80,90 20,60 20,60 80,90 80)),((60 10,10 10,10 90,60 90,60 80, 40 80,40 20,60 20,60 10)))`	`MULTIPOLYGON (((60 90,60 80,90 80,90 20,60 20,60 10, 10 10,10 90,60 90)))`

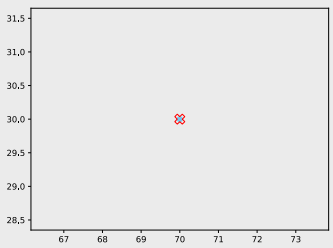
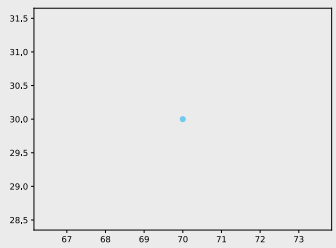
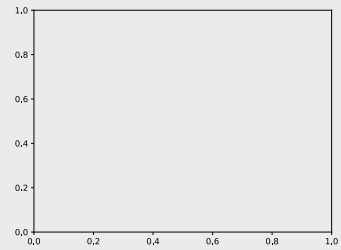
MultiPolygon with multiple overlapping Polygons

Input geometry and result of `gdal vector check-geometry`	`gdal vector make-valid --method=linework`	`gdal vector make-valid --method=structure`
		
Input geometry: `MULTIPOLYGON (((90 90,90 30,30 30,30 90, 90 90)),((20 20,20 80,80 80,80 20,20 20)),((10 10,10 70,70 70,70 10,10 10)))` Error message: `Self-intersection` Error geometry: `MULTIPOINT (70 20)`	`MULTIPOLYGON (((10 10,10 70,20 70,20 20,70 20,70 10, 10 10)),((30 80,30 70,20 70,20 80,30 80)),((90 90, 90 30,80 30,80 80,30 80,30 90,90 90)),((70 20,70 30,80 30,80 20,70 20)),((30 30,30 70,70 70,70 30,30 30)))`	`MULTIPOLYGON (((10 70,20 70,20 80,30 80,30 90,90 90, 90 30,80 30,80 20,70 20,70 10,10 10,10 70)))`

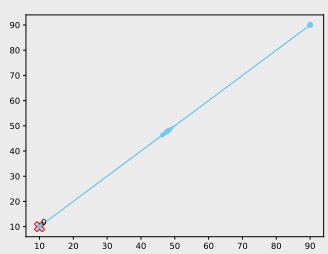
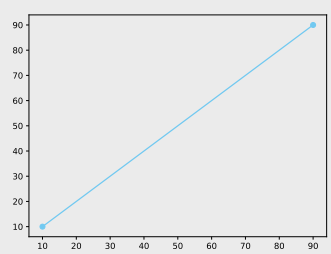
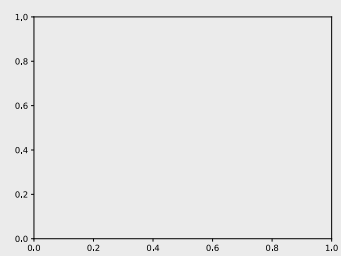
MultiPolygon with two adjacent Polygons

Input geometry and result of `gdal vector check-geometry`	`gdal vector make-valid --method=linework`	`gdal vector make-valid --method=structure`
		
Input geometry: `MULTIPOLYGON (((10 90,50 90,50 10,10 10, 10 90)),((90 80,90 20,50 20,50 80,90 80)))` Error message: `Self-intersection` Error geometry: `MULTIPOINT (50 80)`	`POLYGON ((50 80,90 80,90 20,50 20,50 10,10 10,10 90, 50 90,50 80))`	`MULTIPOLYGON (((50 90,50 80,90 80,90 20,50 20,50 10, 10 10,10 90,50 90)))`

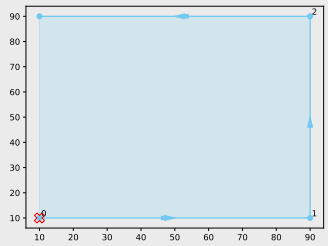
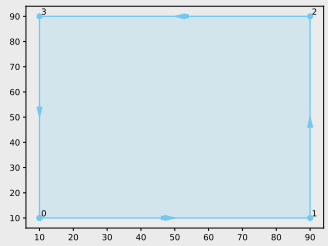
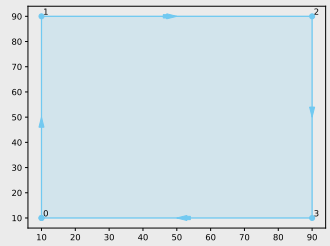
Single-point polygon

Input geometry and result of `gdal vector check-geometry`	`gdal vector make-valid --method=linework`	`gdal vector make-valid --method=structure`
		
Input geometry: `POLYGON ((70 30))` Error message: `point array must contain 0 or >1 elements` Error geometry: `MULTIPOINT (70 30)`	`POINT (70 30)`	`POLYGON EMPTY`

Two-point polygon

Input geometry and result of `gdal vector check-geometry`	`gdal vector make-valid --method=linework`	`gdal vector make-valid --method=structure`
		
Input geometry: `POLYGON ((10 10,90 90))` Error message: `Points of LinearRing do not form a closed linestring` Error geometry: `MULTIPOINT (10 10)`	`LINESTRING (10 10,90 90)`	`POLYGON EMPTY`

Non-closed ring

Input geometry and result of `gdal vector check-geometry`	`gdal vector make-valid --method=linework`	`gdal vector make-valid --method=structure`
		
Input geometry: `POLYGON ((10 10,90 10,90 90,10 90))` Error message: `Points of LinearRing do not form a closed linestring` Error geometry: `MULTIPOINT (10 10)`	`POLYGON ((10 10,90 10,90 90,10 90,10 10))`	`POLYGON ((10 10,10 90,90 90,90 10,10 10))`

8.9 Geographic Networks Data Model

This document is intended to describe the purpose and the structure of Geographic Network Model classes. GNM is the part of GDAL and provides the methods of creating, managing and analysing geographical networks.

The key purpose of GNM classes: - To provide an abstraction for different existed network formats, like GDAL (previously OGR) provides one for spatial vector formats; - To provide a network functionality to those spatial formats which does not have it at all.

8.9.1 General concept

Any real-world network can be represented as a set of vector data, which can be itself represented in GDAL as a GDALDataset. In GNM this data consists of two parts. Network's topology (graph), network's metadata (name/description), set of special feature identifiers, etc. belong to the "network part", while the common for GDAL layers, features, geometries belong to the "spatial/attribute part". In order to work with the datasets of different formats the following classes were designed in GNM.

8.9.1.1 Network

`GNMNetwork` represents an abstract network. The network data and spatial/attribute data in a dataset of some format in fact can be not separable (just additional layers/fields/tags), while the concrete implementation of GNMNetwork "knows" which data from the whole dataset refers to "network part" and is able to operate it. GNMNetwork allows user the following:

-Setting/unsetting connections. These generic methods of building the network topology (automatically and manually) receive the identifiers of features being connected in a common way, while the concrete implementation knows where and how to store and build the topology; -Reading connections. The generic methods return the connections in the common way; -Adding/removing layers/features. When the feature or layer is being added to the network some actions can be initiated (weights change in a graph, cascade changes in connected features). Concrete GNMNetwork describes how it is done. -Defining network's business logic or behavior. It can be expressed in network rules or constraints/restrictions. Expected that each rule can be set from a string and each concrete GNMNetwork will transform it to the internal look.

8.9.1.2 Format

GNMNetwork inherits GDALDataset and looks like OGRDatasource with additional functionality. There are a set of GDAL drivers for networks. The generic network implementation in GDAL provides additional functionality like rules, virtual edges and vertices. Also, while editing the feature the network control the network rules and other specific, and can deny saving edits. The other network drivers (pgRouting, OSRM, GraphHopper, etc.) should provide the basic functionality via the GNMNetwork class.

8.9.2 Network formats

To add a `native` support of the existed network format (like PostGIS pgRouting, Oracle Spatial Networks, topology in GML, etc.) to GNM the developer should implement the corresponding GNMDriver-GNMNetwork interface. But there is also a capability to use the `generic` network format, which is already implemented in GNM as a special class. It can be extremely useful when there is a need to create and use a network in the format that initially does not have its "network part" (like ESRI Shapefile) directly.

8.9.2.1 GNMGenericNetwork

`GNMGenericNetwork` is a concrete implementation of the GNMNetwork. GNMGenericNetwork intends to support the most GDALDataset drivers (depends on the corresponding driver capabilities). Technically the network format abstraction is achieved with the help of GDAL abstraction: datasets and layers approach. GNMGdalNetwork aggregates a GDALDataset instance where the "network part" is represented as a set of "system layers" (wkbNone geometry, specific attribute fields) and the spatial/attribute data is regarded as the set of "class layers" or "classes" (layers with geometries and attributes, as usual). The "network part" is created and maintained by GNMGenericNetwork automatically and provides methods to work with it.

The way of describing real-world networks by GNMGenericNetwork intends to be a generic, because: -The most general type of graph is used, which holds every useful information: directions of edges (directed/undirected), edge costs (weighted/unweighted). This graph is stored as an incidence list: source vertex feature id, target vertex feature id, edge feature id, direct cost, inverse cost, direction of edge; -Any feature with any geometry can be the vertex or the edge in a graph. Also, it may be no feature “under” the connection's edge at all (actually the virtual edge is created for this case). All this means that user operates with the feature identifiers, while the GNMGenericNetwork guaranties the connections integrity among features; -Any feature in the network will gain the unique identifier – Global Feature Identifier (GFID) which allows unify any amount of "class layers" under one network; -GNMGenericNetwork uses its own way to determine the network's business logic. See *`GNMGenericNetwork::CreateRule()`* for more details.

See the *`GNMGenericNetwork`* class documentation for more details.

The network of common format has also the following important features: -The single spatial reference system is used in the network, that means that each feature which appears in the network will be transformed to this SRS; -The network always created void and there is a need to import or create features; -It is not possible to remove the "network part" from the dataset – only delete the whole network with all data. The deletion is made layer by layer and deletes only system and class layers which registered in the network.

8.9.3 Network analysis

The network analysis in GNM is implemented in *`GNMNetwork`* object.

`GNMGenericNetwork` holds the graph in memory in STL containers and provides basic algorithms which return the results in the array-form (e.g. std::vector full of path's edges and vertices GFIDs). But the caller get a result as OGRLayer there features get from layers consist the network. Also some additional fields created (VERTEX/EDGE indicator field, GFID, layer name, etc.). The caller have to free the result OGRLayer via *`GDALDataset::ReleaseResultSet()`*

8.10 Multi-threading

8.10.1 GDAL API: re-entrant, but (generally) not thread-safe

The exact meaning of the terms `thread-safe` or `re-entrant` is not fully standardized. We will use here the QT definitions. In particular, a C function or C++ method is said to be re-entrant if it can be called simultaneously from multiple threads, *but* only if each invocation uses its own data.

All GDAL public C functions and C++ methods are re-entrant, except:

- the general initialization functions, like *GDALAllRegister()*.
- the general cleanup functions like *GDALDestroy()* or *OSRCleanup()*.

Those functions should not be called concurrently from several threads, and it is general best practice to call them from the main thread of the program at program initialization and termination.

Unless otherwise stated, no GDAL public C functions and C++ methods should be assumed to be thread-safe. That is you should not call simultaneously GDAL functions from multiple threads on the same data instance, or even instances that are closely related through ownership relationships. For example, for a multi-band raster dataset, it is not safe to call concurrently GDAL functions on different *GDALRasterBand* instances owned by the same *GDALDataset* instance (each thread should instead manipulate a distinct GDALDataset). Similarly for a GDALDataset owning several *OGRLayer*.

The reason is that most implementations of GDALDataset or GDALRasterBand are stateful. A GDALDataset typically owns a file handle, and performs seek/read operations on it, thus not allowing concurrent access. Block cache related structures for a given GDALDataset are not thread-safe. Drivers also often implement lazy initialization strategies to access various metadata which are resolved only the first time the method to access them is invoked. Drivers may also rely on third-party libraries that expose objects that are not thread-safe.

Those restrictions apply to the C and C++ ABI, and all languages bindings (unless they would take special precautions to serialize calls)

8.10.2 Thread-safe GDAL dataset instances for raster read-only use cases

Added in version 3.10.

rfc-101 adds a new capability to open, or obtain, a thread-safe dataset from any dataset, but only for raster read-only use cases.

At open time, this can be done by passing `GDAL_OF_RASTER | GDAL_OF_THREAD_SAFE` to *GDALOpenEx()* / *GDALDataset::Open()*.

Given an existing GDALDataset* instance, *GDALDataset::IsThreadSafe()* can be used to determine if it is thread-safe or not. If not, *GDALGetThreadSafeDataset()* can be used.

Note that the generic implementation of this capability involves opening one dataset the first time a thread-safe dataset/raster band is accessed by a thread. While this is an implementation detail that can be ignored to develop code, it is important to note regarding potential performance impacts

8.10.3 GDAL block cache and multi-threading

The current design of the GDAL raster block cache allows concurrent reads of several datasets. However performance issues may arise when writing several datasets from several threads, due to lock contention in the global structures of the block cache mechanism.

8.10.4 RAM fragmentation and multi-threading

It has been observed that scenarios that involve multi-threading reading or writing of raster datasets are prone to cause a high RAM usage, in particular when using the default dynamic memory allocator of Linux. Using the alternate tcmalloc memory allocator helps reducing the amount of virtual and resident memory used.

For example, with Debian/Ubuntu distributions, this can be done by installing the `libtcmalloc-minimal4` package and running the binary that executes GDAL with:

```
LD_PRELOAD=/usr/lib/x86_64-linux-gnu/libtcmalloc_minimal.so.4 ./binary
```

8.10.5 GDAL and multi-processing

POSIX fork() API should not be called during the middle of a GDAL operation, otherwise some structures like mutexes might appear to be locked forever in the forked process. If multi-processing is done, we recommend that processes are forked before any GDAL operation is done. Operating on the same GDALDataset instance in several sub-processes will generally lead to wrong results due to the underlying file descriptors being shared.

8.11 OGR SQL dialect and SQLITE SQL dialect

The GDALDataset supports executing commands against a datasource via the *`GDALDataset::ExecuteSQL()`* method. How such commands are evaluated is dependent on the datasets.

- For most file formats (e.g. Shapefiles, GeoJSON, MapInfo files), the built-in *OGR SQL dialect* dialect will be used by defaults. It is also possible to request the *SQL SQLite dialect* alternate dialect to be used, which will use the SQLite engine to evaluate commands on GDAL datasets.
- All OGR drivers for database systems: *MySQL*, *PostgreSQL / PostGIS*, *Oracle Spatial*, *SQLite / Spatialite RDBMS*, *GPKG -- GeoPackage vector*, *ODBC RDBMS*, *ESRI Personal GeoDatabase*, *SAP HANA* and *MSSQLSpatial - Microsoft SQL Server Spatial Database*, override the *`GDALDataset::ExecuteSQL()`* function with dedicated implementation and, by default, pass the SQL statements directly to the underlying RDBMS. In these cases the SQL syntax varies in some particulars from OGR SQL. Also, anything possible in SQL can then be accomplished for these particular databases. Generally, only the result of SELECT statements will be returned as layers. For those drivers, it is also possible to explicitly request the `OGRSQL` and `SQLITE` dialects, although performance will generally be much less as the native SQL engine of those database systems.

8.11.1 Dialects

8.11.1.1 OGR SQL dialect

The GDALDataset supports executing commands against a datasource via the *`GDALDataset::ExecuteSQL()`* method. While in theory any sort of command could be handled this way, in practice the mechanism is used to provide a subset of SQL SELECT capability to applications. This page discusses the generic SQL implementation implemented within OGR, and issues with driver specific SQL support.

The `OGRSQL` dialect can be requested with the `OGRSQL` string passed as the dialect parameter of *`GDALDataset::ExecuteSQL()`*, or with the *-dialect* option switch of the *ogrinfo* or *ogr2ogr* utilities.

An alternate dialect, the `SQLite` dialect, can be used instead of the `OGRSQL` dialect. Refer to the *SQL SQLite dialect* page for more details.

The OGRLayer class also supports applying an attribute query filter to features returned using the *`OGRLayer::SetAttributeFilter()`* method. The syntax for the attribute filter is the same as the WHERE clause in the OGR SQL SELECT statement. So everything here with regard to the WHERE clause applies in the context of the `SetAttributeFilter()` method.

SELECT

The SELECT statement is used to fetch layer features (analogous to table rows in an RDBMS) with the result of the query represented as a temporary layer of features. The layers of the datasource are analogous to tables in an RDBMS and feature attributes are analogous to column values. The simplest form of OGR SQL SELECT statement looks like this:

```
SELECT * FROM polylayer
```

In this case all features are fetched from the layer named "polylayer", and all attributes of those features are returned. This is essentially equivalent to accessing the layer directly. In this example the "*" is the list of fields to fetch from the layer, with "*" meaning that all fields should be fetched.

This slightly more sophisticated form still pulls all features from the layer but the schema will only contain the geometry column and the EAS_ID and PROP_VALUE attributes. With OGR SQL dialect the geometry column is always included in the result so it does not need to appear in the SQL statement.

```
SELECT eas_id, prop_value FROM polylayer
```

A much more ambitious SELECT, restricting the features fetched with a WHERE clause, and sorting the results might look like:

```
SELECT * from polylayer WHERE prop_value > 220000.0 ORDER BY prop_value DESC
```

This select statement will produce a table with just one feature, with geometry and one attribute (named something like "count_eas_id") containing the number of distinct values of the eas_id attribute.

```
SELECT COUNT(DISTINCT eas_id) FROM polylayer
```

General syntax

The general syntax of a SELECT statement is:

```
SELECT [fields] FROM layer_name [JOIN ...] [WHERE ...] [ORDER BY ...] [LIMIT ...]␣
↪[OFFSET ...]
```

List Operators

The field list is a comma-separated list of the fields to be carried into the output features from the source layer. They will appear on output features in the order they appear on in the field list, so the field list may be used to re-order the fields. The special character * is taken to mean "all fields". The syntax `* EXCLUDE ([fields])` can be used to select all fields except those listed in parentheses.

A special form of the field list uses the DISTINCT keyword. This returns a list of all the distinct values of the named attribute. When the DISTINCT keyword is used, only one attribute may appear in the field list. The DISTINCT keyword may be used against any type of field. Currently the distinctness test against a string value is case insensitive in OGR SQL. The result of a SELECT with a DISTINCT keyword is a layer with one column (named the same as the field operated on), and one feature per distinct value. Geometries are discarded. The distinct values are assembled in memory, so a lot of memory may be used for datasets with a large number of distinct values.

```
SELECT DISTINCT areacode FROM polylayer
```

There are also several summarization operators that may be applied to columns. When a summarization operator is applied to any field, then all fields must have summarization operators applied. The summarization operators are:

- COUNT: count of instances

- AVG: numerical average:
- SUM: numerical sum
- MIN: lexical or numerical minimum
- MAX: lexical or numerical maximum
- STDDEV_POP: (GDAL >= 3.10) numerical population standard deviation. Applied on Date/DateTime/Time fields, this returns a value in seconds.
- STDDEV_SAMP: (GDAL >= 3.10) numerical sample standard deviation. Applied on Date/DateTime/Time fields, this returns a value in seconds.

This example produces a variety of summarization information on parcel property values:

```
SELECT MIN(prop_value), MAX(prop_value), AVG(prop_value), SUM(prop_value),
    COUNT(prop_value), STDDEV_POP(prop_value) FROM polylayer WHERE prov_name = 'Ontario'
```

It is also possible to apply the COUNT() operator to a DISTINCT SELECT to get a count of distinct values, for instance:

```
SELECT COUNT(DISTINCT areacode) FROM polylayer
```

As a special case, the COUNT() operator can be given a "*" argument instead of a field name which is a short form for count all the records.

```
SELECT COUNT(*) FROM polylayer
```

Field names can also be prefixed by a table name though this is only really meaningful when performing joins. It is further demonstrated in the JOIN section.

Field definitions can also be complex expressions using arithmetic, and functional operators. However, the DISTINCT keyword, and summarization operators like MIN, MAX, AVG and SUM may not be applied to expression fields. Boolean resulting expressions (comparisons, logical operators) can also be used.

```
SELECT cost+tax from invoice
```

or

```
SELECT CONCAT(owner_first_name,' ',owner_last_name) from properties
```

Functions

The SUBSTR function can be used to extract a substring from a string. Its syntax is the following one : SUBSTR(string_expr, start_offset [, length]). It extracts a substring of string_expr, starting at offset start_offset (1 being the first character of string_expr, 2 the second one, etc...). If start_offset is a negative value, the substring is extracted from the end of the string (-1 is the last character of the string, -2 the character before the last character, ...). If length is specified, up to length characters are extracted from the string. Otherwise the remainder of the string is extracted.

Note: for the time being, the character as considered to be equivalent to bytes, which may not be appropriate for multi-byte encodings like UTF-8.

```
SELECT SUBSTR('abcdef',1,2) FROM xxx   --> 'ab'
SELECT SUBSTR('abcdef',4)   FROM xxx   --> 'def'
SELECT SUBSTR('abcdef',-2)  FROM xxx   --> 'ef'
```

The `hstore_get_value()` function can be used to extract a value associate to a key from a HSTORE string, formatted like 'key=>value,other_key=>other_value,...'

```
SELECT hstore_get_value('a => b, "key with space"=> "value with space"', 'key with space
↪') FROM xxx --> 'value with space'
```

Using the field name alias

OGR SQL supports renaming the fields following the SQL92 specification by using the AS keyword according to the following example:

```
SELECT *, OGR_STYLE AS STYLE FROM polylayer
```

The field name alias can be used as the last operation in the column specification. Therefore we cannot rename the fields inside an operator, but we can rename whole column expression, like these two:

```
SELECT COUNT(areacode) AS "count" FROM polylayer
SELECT dollars*100.0 AS cents FROM polylayer
```

Changing the type of the fields

OGR SQL supports changing the type of the columns by using the SQL92 compliant CAST operator according to the following example:

```
SELECT *, CAST(OGR_STYLE AS character(255)) FROM rivers
```

Currently casting to the following target types are supported:

- boolean
- character(field_length). By default, field_length=1.
- float(field_length)
- numeric(field_length, field_precision)
- smallint(field_length) : 16 bit signed integer
- integer(field_length)
- bigint(field_length), 64 bit integer, extension to SQL92
- date(field_length)
- time(field_length)
- timestamp(field_length)
- geometry, geometry(geometry_type), geometry(geometry_type,epsg_code)

Specifying the field_length and/or the field_precision is optional. An explicit value of zero can be used as the width for character() to indicate variable width. Conversion to the 'integer list', 'double list' and 'string list' OGR data types are not supported, which doesn't conform to the SQL92 specification.

While the CAST operator can be applied anywhere in an expression, including in a WHERE clause, the detailed control of output field format is only supported if the CAST operator is the "outer most" operators on a field in the field definition list. In other contexts it is still useful to convert between numeric, string and date data types.

Casting a WKT string to a geometry is allowed. geometry_type can be POINT[Z], LINESTRING[Z], POLYGON[Z], MULTIPOINT[Z], MULTILINESTRING[Z], MULTIPOLYGON[Z], GEOMETRYCOLLECTION[Z] or GEOMETRY[Z].

String literals and identifiers quoting

Strict SQL92 rules are applied regarding string literals and identifiers quoting.

String literals (constants) must be surrounded with single-quote characters. e.g. WHERE a_field = 'a_value'

Identifiers (column names and tables names) can be used unquoted if they don't contain special characters or are not a SQL reserved keyword. Otherwise they must be surrounded with double-quote characters. e.g. WHERE "from" = 5.

WHERE

The argument to the WHERE clause is a logical expression used select records from the source layer. In addition to its use within the WHERE statement, the WHERE clause handling is also used for OGR attribute queries on regular layers via *OGRLayer::SetAttributeFilter()*.

In addition to the arithmetic and other functional operators available in expressions in the field selection clause of the SELECT statement, in the WHERE context logical operators are also available and the evaluated value of the expression should be logical (true or false).

The available logical operators are =, `!=`, <>, <, >, <=, >=, `LIKE` and `ILIKE`, `BETWEEN` and `IN`. Most of the operators are self explanatory, but it is worth noting that `!=` is the same as <>, the string equality is case insensitive, but the <, >, <= and >= operators *are* case sensitive.

Starting with GDAL 3.1, LIKE is case sensitive, and ILIKE is case insensitive. In previous versions, LIKE was also case insensitive. If the old behavior is wished in GDAL 3.1, the *OGR_SQL_LIKE_AS_ILIKE* can be set to `YES`.

Starting with GDAL 3.9, for layers declaring the OLCStringsAsUTF8 capability (that is the content of their fields of String type is UTF-8 encoded), UTF-8 characters are taken into account by `LIKE` and `ILIKE` operators. For ILIKE case insensitive comparisons, this is restricted to the ASCII, Latin-1 Supplement, Latin Extended-A, Latin Extended-B, Greek and Coptic and Cyrillic Unicode categories.

The value argument to the `LIKE` and `ILIKE` operators is a pattern against which the value string is matched. In this pattern percent (%) matches any number of characters, and underscore (_) matches any one character. An optional ESCAPE escape_char clause can be added so that the percent or underscore characters can be searched as regular characters, by being preceded with the escape_char.

```
String              Pattern       Matches?
------              -------       --------
Alberta             ALB%          Yes
Alberta             _lberta       Yes
St. Alberta         _lberta       No
St. Alberta         %lberta       Yes
Robarts St.         %Robarts%     Yes
12345               123%45        Yes
123.45              12?45         No
N0N 1P0             %N0N%         Yes
L4C 5E2             %N0N%         No
```

The `IN` takes a list of values as its argument and tests the attribute value for membership in the provided set.

```
Value               Value Set            Matches?
------              -------              --------
321                 IN (456,123)         No
'Ontario'           IN ('Ontario','BC')  Yes
'Ont'               IN ('Ontario','BC')  No
1                   IN (0,2,4,6)         No
```

The syntax of the BETWEEN operator is "field_name BETWEEN value1 AND value2" and it is equivalent to "field_name >= value1 AND field_name <= value2".

In addition to the above binary operators, there are additional operators for testing if a field is null or not. These are the IS NULL and IS NOT NULL operators.

Basic field tests can be combined in more complicated predicates using logical operators include AND, OR, and the unary logical NOT. Subexpressions should be bracketed to make precedence clear. Some more complicated predicates are:

```
SELECT * FROM poly WHERE (prop_value >= 100000) AND (prop_value < 200000)
SELECT * FROM poly WHERE NOT (area_code LIKE 'N0N%')
SELECT * FROM poly WHERE (prop_value IS NOT NULL) AND (prop_value < 100000)
```

WHERE Limitations

- Fields must all come from the primary table (the one listed in the FROM clause).
- All string comparisons are case insensitive except for <, >, <= and >=

ORDER BY

The ORDER BY clause is used force the returned features to be reordered into sorted order (ascending or descending) on one or multiple fields. Ascending (increasing) order is the default if neither the ASC or DESC keyword is provided. For example:

```
SELECT * FROM property WHERE class_code = 7 ORDER BY prop_value DESC
SELECT * FROM property ORDER BY prop_value
SELECT * FROM property ORDER BY prop_value ASC
SELECT DISTINCT zip_code FROM property ORDER BY zip_code
SELECT * FROM property ORDER BY prop_value ASC, another_field DESC
```

Note that ORDER BY clauses cause two passes through the feature set. One to build an in-memory table of field values corresponded with feature ids, and a second pass to fetch the features by feature id in the sorted order. For formats which cannot efficiently randomly read features by feature id this can be a very expensive operation.

Sorting of string field values is case sensitive, not case insensitive like in most other parts of OGR SQL.

LIMIT and OFFSET

The LIMIT clause can be used to limit the number of features returned. For example

```
SELECT * FROM poly LIMIT 5
```

The OFFSET clause can be used to skip the first features of the result set. The value after OFFSET is the number of features skipped. For example, to skip the first 3 features from the result set:

```
SELECT * FROM poly OFFSET 3
```

Both clauses can be combined:

```
SELECT * FROM poly LIMIT 5 OFFSET 3
```

JOINs

OGR SQL supports a limited form of one to one JOIN. This allows records from a secondary table to be looked up based on a shared key between it and the primary table being queried. For instance, a table of city locations might include a **nation_id** column that can be used as a reference into a secondary **nation** table to fetch a nation name. A joined query might look like:

```
SELECT city.*, nation.name FROM city
    LEFT JOIN nation ON city.nation_id = nation.id
```

This query would result in a table with all the fields from the city table, and an additional "nation.name" field with the nation name pulled from the nation table by looking for the record in the nation table that has the "id" field with the same value as the city.nation_id field.

Joins introduce a number of additional issues. One is the concept of table qualifiers on field names. For instance, referring to city.nation_id instead of just nation_id to indicate the nation_id field from the city layer. The table name qualifiers may only be used in the field list, and within the `ON` clause of the join.

Wildcards are also somewhat more involved. All fields from the primary table (**city** in this case) and the secondary table (**nation** in this case) may be selected using the usual * wildcard. But the fields of just one of the primary or secondary table may be selected by prefixing the asterisk with the table name.

The field names in the resulting query layer will be qualified by the table name, if the table name is given as a qualifier in the field list. In addition field names will be qualified with a table name if they would conflict with earlier fields. For instance, the following select would result might result in a results set with a **name, nation_id, nation.nation_id** and ** nation.name** field if the city and nation tables both have the **nation_id** and **name** fieldnames.

```
SELECT * FROM city LEFT JOIN nation ON city.nation_id = nation.nation_id
```

On the other hand if the nation table had a **continent_id** field, but the city table did not, then that field would not need to be qualified in the result set. However, if the selected instead looked like the following statement, all result fields would be qualified by the table name.

```
SELECT city.*, nation.* FROM city
    LEFT JOIN nation ON city.nation_id = nation.nation_id
```

In the above examples, the **nation** table was found in the same datasource as the **city** table. However, the OGR join support includes the ability to join against a table in a different data source, potentially of a different format. This is indicated by qualifying the secondary table name with a datasource name. In this case the secondary datasource is opened using normal OGR semantics and utilized to access the secondary table until the query result is no longer needed.

```
SELECT * FROM city
LEFT JOIN '/usr2/data/nation.dbf'.nation ON city.nation_id = nation.nation_id
```

While not necessarily very useful, it is also possible to introduce table aliases to simplify some SELECT statements. This can also be useful to disambiguate situations where tables of the same name are being used from different data sources. For instance, if the actual tables names were messy we might want to do something like:

```
SELECT c.name, n.name FROM project_615_city c
LEFT JOIN '/usr2/data/project_615_nation.dbf'.project_615_nation n
            ON c.nation_id = n.nation_id
```

It is possible to do multiple joins in a single query.

```
SELECT city.name, prov.name, nation.name FROM city
LEFT JOIN province ON city.prov_id = province.id
LEFT JOIN nation ON city.nation_id = nation.id
```

The expression after ON is typically of the form "{primary_table}.{field_name} = {secondary_table}.{field_name}", and in that order. It is also possible to use a more complex boolean expression, involving multiple comparison operators, but with the restrictions mentioned in the below "JOIN limitations" section. In particular, in case of multiple joins (3 tables or more) the fields compared in a JOIN must belong to the primary table (the one after FROM) and the table of the active JOIN.

JOIN Limitations

- Joins can be very expensive operations if the secondary table is not indexed on the key field being used.
- Joined fields may not be used in WHERE clauses, or ORDER BY clauses at this time. The join is essentially evaluated after all primary table subsetting is complete, and after the ORDER BY pass.
- Joined fields may not be used as keys in later joins. So you could not use the province id in a city to lookup the province record, and then use a nation id from the province id to lookup the nation record. This is a sensible thing to want and could be implemented, but is not currently supported.
- Datasource names for joined tables are evaluated relative to the current processes working directory, not the path to the primary datasource.
- These are not true LEFT or RIGHT joins in the RDBMS sense. Whether or not a secondary record exists for the join key or not, one and only one copy of the primary record is returned in the result set. If a secondary record cannot be found, the secondary derived fields will be NULL. If more than one matching secondary field is found only the first will be used.

UNION ALL

The SQL engine can deal with several SELECT combined with UNION ALL. The effect of UNION ALL is to concatenate the rows returned by the right SELECT statement to the rows returned by the left SELECT statement.

```
[(] SELECT field_list FROM first_layer [WHERE where_expr] [)]
UNION ALL [(] SELECT field_list FROM second_layer [WHERE where_expr] [)]
[UNION ALL [(] SELECT field_list FROM third_layer [WHERE where_expr] [)]]*
```

UNION ALL restrictions

The processing of UNION ALL in OGR differs from the SQL standard, in which it accepts that the columns from the various SELECT are not identical. In that case, it will return a super-set of all the fields from each SELECT statement.

There is also a restriction : ORDER BY can only be specified for each SELECT, and not at the level of the result of the union.

SPECIAL FIELDS

The OGR SQL query processor treats some attributes of the features as built-in special fields which can be used in SQL statements alongside the other fields. These fields can be placed in the SELECT list, the WHERE clause and the ORDER BY clause respectively. Special fields will not be included in the result by default but they may be explicitly included by adding them to the SELECT list. When accessing field values, the special fields will take precedence over other fields in the data source with the same names.

Feature id (FID)

Normally the feature id is a special property of a feature and not treated as an attribute of the feature. In some cases it is convenient to be able to utilize the feature id in queries and result sets as a regular field. To do so use the name `FID`. If the layer has a named FID column (*OGRLayer::GetFIDColumn()* != ""), this name may also be used.

The field wildcard expansions will not include the feature id, but it may be explicitly included using a syntax like:

```
SELECT FID, * FROM nation
```

Geometry field

The OGR SQL dialect adds the geometry field of the datasource to the result set by default. Users do not need to select the geometry explicitly but it is still possible to do so. Common use case is when geometry is the only field that is needed. In this case the name of the geometry field to be used in the SQL statement is the name returned by *OGRLayer::GetGeometryColumn()*, and also "Geometry Column = ..." in **ogrinfo** output. If the method returns an empty string then a special name "_ogr_geometry_" must be used. The name begins with an underscore and SQL syntax requires that it must appear between double quotes. In addition the command line interpreter may require that double quotes are escaped and the final SELECT statement could look like:

```
SELECT "_ogr_geometry_" FROM nation
```

OGR_GEOMETRY

Some of the data sources (like MapInfo tab) can handle geometries of different types within the same layer. The `OGR_GEOMETRY` special field represents the geometry type returned by *OGRGeometry::getGeometryName()* and can be used to distinguish the various types. By using this field one can select particular types of the geometries like:

```
SELECT * FROM nation WHERE OGR_GEOMETRY='POINT' OR OGR_GEOMETRY='POLYGON'
```

OGR_GEOM_WKT

The Well Known Text representation of the geometry can also be used as a special field. To select the WKT of the geometry `OGR_GEOM_WKT` might be included in the select list, like:

```
SELECT OGR_GEOM_WKT, * FROM nation
```

Using the `OGR_GEOM_WKT` and the `LIKE` operator in the WHERE clause we can get similar effect as using OGR_GEOMETRY:

```
SELECT OGR_GEOM_WKT, * FROM nation WHERE OGR_GEOM_WKT
LIKE 'POINT%' OR OGR_GEOM_WKT LIKE 'POLYGON%'
```

OGR_GEOM_AREA

The `OGR_GEOM_AREA` special field returns the area of the feature's geometry computed by the OGRSurface::get_Area() method. For OGRGeometryCollection and OGRMultiPolygon the value is the sum of the areas of its members. For non-surface geometries the returned area is 0.0.

For example, to select only polygon features larger than a given area:

```
SELECT * FROM nation WHERE OGR_GEOM_AREA > 10000000
```

OGR_STYLE

The OGR_STYLE special field represents the style string of the feature returned by OGRFeature::GetStyleString(). By using this field and the LIKE operator the result of the query can be filtered by the style. For example we can select the annotation features as:

```
SELECT * FROM nation WHERE OGR_STYLE LIKE 'LABEL%'
```

It is possible to use the OGR_STYLE field name as a special field name in the field selection as an alternate way of setting the *OGRFeature::SetStyleString()* value, typically by aliasing another field or a string literal.

```
SELECT *, 'BRUSH(fc:#01234567)' AS OGR_STYLE FROM source_layer
```

By default, the OGR_STYLE field will still be visible as a regular field. If this is undesirable, starting with GDAL 3.10, it can be hidden by adding the HIDDEN keyword at the end of the field specification.

```
SELECT * EXCLUDE(my_style_field), my_style_field AS OGR_STYLE HIDDEN FROM source_layer
```

CREATE INDEX

Some OGR SQL drivers support creating of attribute indexes. Currently this includes the Shapefile driver. An index accelerates very simple attribute queries of the form **fieldname = value**, which is what is used by the JOIN capability. To create an attribute index on the nation_id field of the nation table a command like this would be used:

```
CREATE INDEX ON nation USING nation_id
```

Index Limitations

- Indexes are not maintained dynamically when new features are added to or removed from a layer.
- Very long strings (longer than 256 characters?) cannot currently be indexed.
- To recreate an index it is necessary to drop all indexes on a layer and then recreate all the indexes.
- Indexes are not used in any complex queries. Currently the only query the will accelerate is a simple "field = value" query.

DROP INDEX

The OGR SQL DROP INDEX command can be used to drop all indexes on a particular table, or just the index for a particular column.

```
DROP INDEX ON nation USING nation_id
DROP INDEX ON nation
```

ALTER TABLE

The following OGR SQL ALTER TABLE commands can be used.

- "ALTER TABLE tablename ADD [COLUMN] columnname columntype" to add a new field. Supported if the layer declares the OLCCreateField capability.
- "ALTER TABLE tablename RENAME [COLUMN] oldcolumnname TO newcolumnname" to rename an existing field. Supported if the layer declares the OLCAlterFieldDefn capability.
- "ALTER TABLE tablename ALTER [COLUMN] columnname TYPE columntype" to change the type of an existing field. Supported if the layer declares the OLCAlterFieldDefn capability.

- "ALTER TABLE tablename DROP [COLUMN] columnname" to delete an existing field. Supported if the layer declares the OLCDeleteField capability.

The columntype value follows the syntax of the types supported by the CAST operator described above.

```
ALTER TABLE nation ADD COLUMN myfield integer
ALTER TABLE nation RENAME COLUMN myfield TO myfield2
ALTER TABLE nation ALTER COLUMN myfield2 TYPE character(15)
ALTER TABLE nation DROP COLUMN myfield2
```

DROP TABLE

The OGR SQL DROP TABLE command can be used to delete a table. This is only supported on datasources that declare the ODsCDeleteLayer capability.

```
DROP TABLE nation
```

8.11.1.2 SQL SQLite dialect

The `SQLite` dialect can be used as an alternate SQL dialect to the *OGR SQL dialect*. This assumes that GDAL/OGR is built with support for SQLite, and preferably with Spatialite support too to benefit from spatial functions.

The SQLite dialect may be used with any OGR datasource, like the OGR SQL dialect. The `SQLite` dialect can be requested with the `SQLite` string passed as the dialect parameter of *`GDALDataset::ExecuteSQL()`*, or with the *-dialect* option of the *ogrinfo* or *ogr2ogr* utilities.

This is mainly aimed to execute SELECT statements, but, for datasources that support update, INSERT/UPDATE/DELETE statements can also be run. GDAL is internally using the Virtual Table Mechanism of SQLite and therefore operations like ALTER TABLE are not supported. For executing ALTER TABLE or DROP TABLE use *OGR SQL dialect*

If the datasource is SQLite database (GeoPackage, SpatiaLite) then SQLite dialect acts as native SQL dialect and Virtual Table Mechanism is not used. It is possible to force GDAL to use Virtual Tables even in this case by specifying "-dialect INDIRECT_SQLITE". This should be used only when necessary, since going through the virtual table mechanism might affect performance.

The syntax of the SQL statements is fully the one of the SQLite SQL engine. You can refer to the following pages:

- SELECT
- INSERT
- UPDATE
- DELETE

SELECT statement

The SELECT statement is used to fetch layer features (analogous to table rows in an RDBMS) with the result of the query represented as a temporary layer of features. The layers of the datasource are analogous to tables in an RDBMS and feature attributes are analogous to column values. The simplest form of OGR SQLITE SELECT statement looks like this:

```
SELECT * FROM polylayer
```

More complex statements can of course be used, including WHERE, JOIN, USING, GROUP BY, ORDER BY, sub SELECT, ...

The table names that can be used are the layer names available in the datasource on which the ExecuteSQL() method is called.

Similarly to OGRSQL, it is also possible to refer to layers of other datasources with the following syntax : "other_datasource_name"."layer_name".

```
SELECT p.*, NAME FROM poly p JOIN "idlink.dbf"."idlink" il USING (eas_id)
```

If the master datasource is SQLite database (GeoPackage, SpatiaLite) it is necessary to use indirect SQLite dialect. Otherwise additional datasources are never opened but tables to be used in joins are searched from the master database.

```
ogrinfo jointest.gpkg -dialect INDIRECT_SQLITE -sql \
"SELECT a.ID,b.ID FROM jointest a JOIN \"jointest2.shp\".\"jointest2\" b ON a.ID=b.ID"
```

The column names that can be used in the result column list, in WHERE, JOIN, ... clauses are the field names of the layers. Expressions, SQLite functions, spatial functions, etc... can also be used.

The conditions on fields expressed in WHERE clauses, or in JOINs are translated, as far as possible, as attribute filters that are applied on the underlying OGR layers. Joins can be very expensive operations if the secondary table is not indexed on the key field being used.

LIKE operator

In SQLite, the LIKE operator is case insensitive, unless `PRAGMA case_sensitive_like = 1` has been issued.

Starting with GDAL 3.9, GDAL installs a custom LIKE comparison, such that UTF-8 characters are taken into account by `LIKE` operator.

For case insensitive comparisons, this is restricted to the ASCII, Latin-1 Supplement, Latin Extended-A, Latin Extended-B, Greek and Coptic and Cyrillic Unicode categories.

Delimited identifiers

If names of layers or attributes are reserved keywords in SQL like 'FROM' or they begin with a number or underscore they must be handled as "delimited identifiers" and enclosed between double quotation marks in queries. Double quotes can be used even when they are not strictly needed.

```
SELECT "p"."geometry", "p"."FROM", "p"."3D" FROM "poly" p
```

When SQL statements are used in the command shell and the statement itself is put between double quotes, the internal double quotes must be escaped with \

```
ogrinfo p.shp -sql "SELECT geometry \"FROM\", \"3D\" FROM p"
```

Geometry field

Geometry fields can be explicitly specified in the result column list of a SELECT, or automatically selected if the * wildcard is used.

For OGR layers that have a non-empty geometry column name (generally for RDBMS datasources), as returned by OGRLayer::GetGeometryColumn(), the name of the geometry special field in the SQL statement must be the name of the geometry column of the underlying OGR layer. If the name of the geometry column in the source layer is empty, like with shapefiles etc., the name to use in the SQL statement must be "geometry". Here we'll use it case-insensitively (as all field names are in a SELECT statement):

```
SELECT EAS_ID, GEOMETRY FROM poly
```

returns:

```
OGRFeature(SELECT):0
EAS_ID (Real) = 168
POLYGON ((479819.84375 4765180.5,479690.1875 4765259.5,[...],479819.84375 4765180.5))
```

```
SELECT * FROM poly
```

returns:

```
OGRFeature(SELECT):0
AREA (Real) = 215229.266
EAS_ID (Real) = 168
PRFEDEA (String) = 35043411
POLYGON ((479819.84375 4765180.5,479690.1875 4765259.5,[...],479819.84375 4765180.5))
```

Feature id (FID)

The feature id is a special property of a feature and not treated as an attribute of the feature. In some cases it is convenient to be able to utilize the feature id in queries and result sets as a regular field. To do so use the name `rowid`.

Starting with GDAL 3.8, if the layer has a named FID column (*`OGRLayer::GetFIDColumn()`* != ""), this name may also be used.

The field wildcard expansions will not include the feature id, but it may be explicitly included using a syntax like:

```
SELECT rowid, * FROM nation
```

It is of course possible to rename it:

```
SELECT rowid AS fid, * FROM nation
```

OGR_STYLE special field

The `OGR_STYLE` special field represents the style string of the feature returned by OGRFeature::GetStyleString(). By using this field and the `LIKE` operator the result of the query can be filtered by the style. For example we can select the annotation features as:

```
SELECT * FROM nation WHERE OGR_STYLE LIKE 'LABEL%'
```

Statistics functions

In addition to standard COUNT(), SUM(), AVG(), MIN(), MAX(), the following aggregate functions are available:

- `STDDEV_POP(numeric_value)`: (GDAL >= 3.10) numerical population standard deviation.
- `STDDEV_SAMP(numeric_value)`: (GDAL >= 3.10) numerical sample standard deviation

Ordered-set aggregate functions

The following aggregate functions are available. Note that they require to allocate an amount of memory proportional to the number of selected rows (for `MEDIAN`, `PERCENTILE` and `PERCENTILE_CONT`) or to the number of values (for `MODE`).

- `MEDIAN(numeric_value)`: (GDAL >= 3.10) (continuous) median (equivalent to `PERCENTILE(numeric_value, 50)`). NULL values are ignored.

- `PERCENTILE(numeric_value, percentage)`: (GDAL >= 3.10) (continuous) percentile, with percentage between 0 and 100 (equivalent to `PERCENTILE_CONT(numeric_value, percentage / 100)`). NULL values are ignored.
- `PERCENTILE_CONT(numeric_value, fraction)`: (GDAL >= 3.10) (continuous) percentile, with fraction between 0 and 1. NULL values are ignored.
- `MODE(value)`: (GDAL >= 3.10): mode, i.e. most frequent input value (strings and numeric values are supported), arbitrarily choosing the first one if there are multiple equally-frequent results. NULL values are ignored.

Spatialite SQL functions

When GDAL/OGR is build with support for the Spatialite library, a lot of extra SQL functions, in particular spatial functions, can be used in results column fields, WHERE clauses, etc....

```
SELECT EAS_ID, ST_Area(GEOMETRY) AS area FROM poly WHERE
    ST_Intersects(GEOMETRY, BuildCircleMbr(479750.6875,4764702.0,100))
```

returns:

```
OGRFeature(SELECT):0
EAS_ID (Real) = 169
area (Real) = 101429.9765625

OGRFeature(SELECT):1
EAS_ID (Real) = 165
area (Real) = 596610.3359375

OGRFeature(SELECT):2
EAS_ID (Real) = 170
area (Real) = 5268.8125
```

Note that due to the loose typing mechanism of SQLite, if a geometry expression returns a NULL value for the first row, this will generally cause OGR not to recognize the column as a geometry column. It might be then useful to sort the results by making sure that non-null geometries are returned first:

```
ogrinfo test.shp -sql "SELECT * FROM (SELECT ST_Buffer(geometry,5) AS geometry FROM␣
↪test) ORDER BY geometry IS NULL ASC" -dialect sqlite
```

Other spatial functions

Since GDAL 3.13, `ST_Hilbert` is available. It encodes a (x, y) pair as the Hilbert curve index (32-bit unsigned integer), for a curve covering the given bounding box.

Four variants are available:

- `ST_Hilbert(x, y, min_x, min_y, max_x, max_y)`, where (x, y) is the point of interest, and (min_x, min_y, max_x, max_y) a bounding box that contains the point.
- `ST_Hilbert(x, y, layer_name)`, where (x, y) is the point of interest, and `layer_name` the name of a layer whose extent is used as the bounding box for the computation.
- `ST_Hilbert(geometry, min_x, min_y, max_x, max_y)`, where geometry is a geometry, and (min_x, min_y, max_x, max_y) a bounding box that contains the geometry. The center of the bounding box of the geometry is used as the point to encode.

- `ST_Hilbert(geometry, layer_name)`, where geometry is a geometry, and `layer_name` the name of a layer whose extent is used as the bounding box for the computation. The center of the bounding box of the geometry is used as the point to encode.

This is typically used to spatially sort a set of geometries. Note that the *gdal vector sort* program can also be used for that purpose.

```
gdal vector sql --sql="SELECT * FROM cities ORDER BY ST_Hilbert(geometry, 'cities')" \
                --dialect=SQLite --input=cities.shp --output=sorted_cities.shp
```

or

```
gdal vector sort --method=hilbert --input=cities.shp --output=sorted_cities.shp
```

OGR datasource SQL functions

The `ogr_datasource_load_layers(datasource_name[, update_mode[, prefix]])` function can be used to automatically load all the layers of a datasource as *VirtualOGR tables*.

```
sqlite> SELECT load_extension('libgdal.so');

sqlite> SELECT load_extension('mod_spatialite');

sqlite> SELECT ogr_datasource_load_layers('poly.shp');
1
sqlite> SELECT * FROM sqlite_master;
table|poly|poly|0|CREATE VIRTUAL TABLE "poly" USING VirtualOGR('poly.shp', 0, 'poly')
```

OGR layer SQL functions

The following SQL functions are available and operate on a layer name : `ogr_layer_Extent()`, `ogr_layer_SRID()`, `ogr_layer_GeometryType()` and `ogr_layer_FeatureCount()`

```
SELECT ogr_layer_Extent('poly'), ogr_layer_SRID('poly') AS srid,
    ogr_layer_GeometryType('poly') AS geomtype, ogr_layer_FeatureCount('poly') AS count
```

```
OGRFeature(SELECT):0
srid (Integer) = 40004
geomtype (String) = POLYGON
count (Integer) = 10
POLYGON ((478315.53125 4762880.5,481645.3125 4762880.5,481645.3125 4765610.5,478315.
↪53125 4765610.5,478315.53125 4762880.5))
```

OGR compression functions

`ogr_deflate(text_or_blob[, compression_level])` returns a binary blob compressed with the ZLib deflate algorithm. See *CPLZLibDeflate()*

`ogr_inflate(compressed_blob)` returns the decompressed binary blob, from a blob compressed with the ZLib deflate algorithm. If the decompressed binary is a string, use CAST(ogr_inflate(compressed_blob) AS VARCHAR). See CPLZLibInflate().

Other functions

The `hstore_get_value()` function can be used to extract a value associate to a key from a HSTORE string, formatted like "key=>value,other_key=>other_value,..."

```
SELECT hstore_get_value('a => b, "key with space"=> "value with space"', 'key with space
↪') --> 'value with space'
```

Raster related functions

The `gdal_get_pixel_value()` function (added in GDAL 3.7) can be used to extract the value of a pixel in a GDAL dataset. It requires the configuration option OGR_SQLITE_ALLOW_EXTERNAL_ACCESS to be set to YES (for security reasons).

It takes 5 or 6 arguments:

- a string with the dataset name
- a band number (numbering starting at 1)
- a string being "georef" to indicate that subsequent values will be georeferenced coordinates, or "pixel" to indicate that subsequent values will be in column, line pixel space
- georeferenced X value or column number
- georeferenced Y value or line number
- resampling method among `nearest` (default), `bilinear`, `cubic`, `cubicspline`. Optional, added in GDAL 3.10

```
SELECT gdal_get_pixel_value('../gcore/data/byte.tif', 1, 'georef', 440720, 3751320)
SELECT gdal_get_pixel_value('../gcore/data/byte.tif', 1, 'pixel', 0, 0)
SELECT gdal_get_pixel_value('../gcore/data/byte.tif', 1, 'pixel', 0.5, 0.5, 'bilinear') ␣
↪-- GDAL >= 3.10
```

OGR geocoding functions

The following SQL functions are available : `ogr_geocode(...)` and `ogr_geocode_reverse(...)`.

`ogr_geocode(name_to_geocode [, field_to_return [, option1 [, option2, ...]]])` where name_to_geocode is a literal or a column name that must be geocoded. field_to_return if specified can be "geometry" for the geometry (default), or a field name of the layer returned by `OGRGeocode()`. The special field "raw" can also be used to return the raw response (XML string) of the geocoding service. option1, option2, etc.. must be of the key=value format, and are options understood by `OGRGeocodeCreateSession()` or OGRGeocode().

This function internally uses the OGRGeocode() API. Refer to it for more details.

```
SELECT ST_Centroid(ogr_geocode('Paris'))
```

returns:

```
OGRFeature(SELECT):0
POINT (2.34287687375113 48.856622357411)
```

```
ogrinfo cities.csv -dialect sqlite -sql "SELECT *, ogr_geocode(city, 'country_code') AS␣
↪country_code, ST_Centroid(ogr_geocode(city)) FROM cities"
```

returns:

```
OGRFeature(SELECT):0
  city (String) = Paris
  country_code (String) = fr
  POINT (2.34287687375113 48.856622357411)

OGRFeature(SELECT):1
  city (String) = London
  country_code (String) = gb
  POINT (-0.109415723431508 51.5004964757441)

OGRFeature(SELECT):2
  city (String) = Rennes
  country_code (String) = fr
  POINT (-1.68185479486048 48.1116771631195)

OGRFeature(SELECT):3
  city (String) = New York
  country_code (String) = us
  POINT (-73.9388908443975 40.6632061220125)

OGRFeature(SELECT):4
  city (String) = Beijing
  country_code (String) = cn
  POINT (116.3912972 39.9057136)
```

`ogr_geocode_reverse(longitude, latitude, field_to_return [, option1 [, option2, ...]])` where longitude, latitude is the coordinate to query. field_to_return must be a field name of the layer returned by OGRGeocodeReverse() (for example 'display_name'). The special field "raw" can also be used to return the raw response (XML string) of the geocoding service. option1, option2, etc.. must be of the key=value format, and are options understood by OGRGeocodeCreateSession() or OGRGeocodeReverse().

`ogr_geocode_reverse(geometry, field_to_return [, option1 [, option2, ...]])` is also accepted as an alternate syntax where geometry is a (Spatialite) point geometry.

This function internally uses the `OGRGeocodeReverse()` API. Refer to it for more details.

Spatialite spatial index

Spatialite spatial index mechanism can be triggered by making sure a spatial index virtual table is mentioned in the SQL (of the form idx_layername_geometrycolumn), or by using the more recent SpatialIndex from the VirtualSpatialIndex extension. In which case, a in-memory RTree will be built to be used to speed up the spatial queries.

For example, a spatial intersection between 2 layers, by using a spatial index on one of the layers to limit the number of actual geometry intersection computations :

```
SELECT city_name, region_name FROM cities, regions WHERE
    ST_Area(ST_Intersection(cities.geometry, regions.geometry)) > 0 AND
    regions.rowid IN (
        SELECT pkid FROM idx_regions_geometry WHERE
            xmax >= MbrMinX(cities.geometry) AND xmin <= MbrMaxX(cities.geometry) AND
            ymax >= MbrMinY(cities.geometry) AND ymin <= MbrMaxY(cities.geometry))
```

or more elegantly :

```
SELECT city_name, region_name FROM cities, regions WHERE
    ST_Area(ST_Intersection(cities.geometry, regions.geometry)) > 0 AND
    regions.rowid IN (
        SELECT rowid FROM SpatialIndex WHERE
            f_table_name = 'regions' AND search_frame = cities.geometry)
```

8.11.2 ExecuteSQL()

SQL is executed against an GDALDataset, not against a specific layer. The call looks like this:

```
OGRLayer * GDALDataset::ExecuteSQL( const char *pszSQLCommand,
                                    OGRGeometry *poSpatialFilter,
                                    const char *pszDialect );
```

The `pszDialect` argument is in theory intended to allow for support of different command languages against a provider, but for now applications should always pass an empty (not NULL) string to get the default dialect.

The `poSpatialFilter` argument is a geometry used to select a bounding rectangle for features to be returned in a manner similar to the *`OGRLayer::SetSpatialFilter()`* method. It may be NULL for no special spatial restriction.

The result of an ExecuteSQL() call is usually a temporary OGRLayer representing the results set from the statement. This is the case for a SELECT statement for instance. The returned temporary layer should be released with *`GDALDataset::ReleaseResultSet()`* method when no longer needed. Failure to release it before the datasource is destroyed may result in a crash.

8.12 GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...

8.12.1 Introduction

GDAL can access files located on "standard" file systems, i.e. in the / hierarchy on Unix-like systems or in C:, D:, etc... drives on Windows. But most GDAL raster and vector drivers use a GDAL-specific abstraction to access files. This makes it possible to access less standard types of files, such as in-memory files, compressed files (.zip, .gz, .tar, .tar.gz archives), encrypted files, standard input and output (STDIN, STDOUT), files stored on network (either publicly accessible, or in private buckets of commercial cloud storage services), etc.

Each special file system has a prefix, and the general syntax to name a file is /vsiPREFIX/...

Example:

```
gdalinfo /vsizip/my.zip/my.tif
```

The leading vsi in /vsiPREFIX/ stands for Virtual System Interface.

8.12.2 Chaining

It is possible to chain multiple file system handlers.

```
# ogrinfo a shapefile in a zip file on the internet:

ogrinfo -ro -al -so /vsizip//vsicurl/https://raw.githubusercontent.com/OSGeo/gdal/master/
↪autotest/ogr/data/shp/poly.zip

# ogrinfo a shapefile in a zip file on an ftp:
```

(continues on next page)

(continued from previous page)

```
ogrinfo -ro -al -so /vsizip//vsicurl/ftp://user:password@example.com/foldername/file.zip/
↪example.shp
```

(Note is also OK to say /vsizip/vsicurl/... with a single slash. (But when writing documentation please still use two.))

8.12.3 Drivers supporting virtual file systems

Virtual file systems can only be used with GDAL or OGR drivers supporting the "large file API", which is now the vast majority of file based drivers. The full list of these formats can be obtained by looking at the driver marked with 'v' when running either `gdalinfo --formats` or `ogrinfo --formats`.

A notable exception is the HDF4 driver.

8.12.4 /vsizip/ (.zip archives)

8.12.4.1 Read capabilities

/vsizip/ is a file handler that allows reading ZIP archives on-the-fly without decompressing them beforehand.

To point to a file inside a zip file, the filename must be of the form `/vsizip/path/to/the/file.zip/path/inside/the/zip/file`, where `path/to/the/file.zip` is relative or absolute and `path/inside/the/zip/file` is the relative path to the file inside the archive.

To use the .zip as a directory, you can use `/vsizip/path/to/the/file.zip` or `/vsizip/path/to/the/file.zip/subdir`. Directory listing is available with *VSIReadDir()*. A *VSIStatL()* ("/vsizip/...") call will return the uncompressed size of the file. Directories inside the ZIP file can be distinguished from regular files with the VSI_ISDIR(stat.st_mode) macro as for regular file systems. Getting directory listing and file statistics are fast operations.

Note: in the particular case where the .zip file contains a single file located at its root, just mentioning `/vsizip/path/to/the/file.zip` will work.

The following configuration options are specific to the /zip/ handler:

- **CPL_SOZIP_ENABLED=[YES/NO/AUTO]:** (GDAL >= 3.7) Defaults to `AUTO`. Determines whether the SOZip optimization should be enabled. If `AUTO`, SOZip will be enabled for uncompressed files larger than *CPL_SOZIP_MIN_FILE_SIZE*.
- **CPL_SOZIP_MIN_FILE_SIZE=value:** (GDAL >= 3.7) Defaults to `1M`. Determines the minimum file size for SOZip to be automatically enabled.

Examples:

```
/vsizip/my.zip/my.tif  (relative path to the .zip)
/vsizip//home/even/my.zip/subdir/my.tif  (absolute path to the .zip)
/vsizip/c:\users\even\my.zip\subdir\my.tif
/vsizip/{/vsizip/my.zip/subdir/subzip.zip}/subdir_in_subzip/my.shp  (alternate syntax␣
↪for a nested .zip)
```

.kmz, .ods and .xlsx extensions are also detected as valid extensions for zip-compatible archives.

An alternate syntax is available so as to enable chaining and not being dependent on .zip extension, e.g.: `/vsizip/{/path/to/the/archive}/path/inside/the/zip/file`. Note that `/path/to/the/archive` may also itself use this alternate syntax.

8.12.4.2 Write capabilities

Write capabilities are also available. They allow creating a new zip file and adding new files to an already existing (or just created) zip file.

Creation of a new zip file:

```
fmain = VSIFOpenL("/vsizip/my.zip", "wb");
subfile = VSIFOpenL("/vsizip/my.zip/subfile", "wb");
VSIFWriteL("Hello World", 1, strlen("Hello world"), subfile);
VSIFCloseL(subfile);
VSIFCloseL(fmain);
```

Addition of a new file to an existing zip:

```
newfile = VSIFOpenL("/vsizip/my.zip/newfile", "wb");
VSIFWriteL("Hello World", 1, strlen("Hello world"), newfile);
VSIFCloseL(newfile);
```

The *GDAL_NUM_THREADS* configuration option can be set to an integer or `ALL_CPUS` to enable multi-threaded compression of a single file. This is similar to the pigz utility in independent mode. By default the input stream is split into 1 MB chunks (the chunk size can be tuned with the *CPL_VSIL_DEFLATE_CHUNK_SIZE* configuration option, with values like "x K" or "x M"), and each chunk is independently compressed (and terminated by a nine byte marker 0x00 0x00 0xFF 0xFF 0x00 0x00 0x00 0xFF 0xFF, signaling a full flush of the stream and dictionary, enabling potential independent decoding of each chunk). This slightly reduces the compression rate, so very small chunk sizes should be avoided. Starting with GDAL 3.7, this technique is reused to generate .zip files following *SOZip (Seek-Optimized ZIP)*.

Read and write operations cannot be interleaved. The new zip must be closed before being re-opened in read mode.

8.12.4.3 SOZip (Seek-Optimized ZIP)

GDAL (>= 3.7) has full read and write support for .zip files following the SOZip (Seek-Optimized ZIP) profile.

- The `/vsizip/` virtual file system uses the SOZip index to perform fast random access within a compressed SOZip-enabled file.
- The *ESRI Shapefile / DBF* and *GPKG -- GeoPackage vector* drivers can directly generate SOZip-enabled .shz/.shp.zip or .gpkg.zip files.
- The *CPLAddFileInZip()* C function, which can compress a file and add it to an new or existing ZIP file, enables the SOZip optimization when relevant (ie when a file to be compressed is larger than 1 MB). SOZip optimization can be forced by setting the *CPL_SOZIP_ENABLED* configuration option to YES. Or totally disabled by setting it to NO.
- The *VSIGetFileMetadata()* method can be called on a filename of the form `/vsizip/path/to/the/file.zip/path/inside/the/zip/file` and with domain = "ZIP" to get information if a SOZip index is available for that file.
- The *sozip* new command line utility can be used to create a seek-optimized ZIP file, to append files to an existing ZIP file, list the contents of a ZIP file and display the SOZip optimization status or validate a SOZip file.

8.12.5 /vsigzip/ (gzipped file)

/vsigzip/ is a file handler that allows on-the-fly reading of GZip (.gz) files without decompressing them in advance.

To view a gzipped file as uncompressed by GDAL, you must use the `/vsigzip/path/to/the/file.gz` syntax, where `path/to/the/file.gz` is relative or absolute.

The following configuration options are specific to the /vsigzip/ handler:

- **CPL_VSIL_GZIP_WRITE_PROPERTIES=[YES/NO]:** Defaults to YES. If YES, when the file is located in a writable location, a file with extension .gz.properties is created with an indication of the uncompressed file size.

Examples:

```
/vsigzip/my.gz # (relative path to the .gz)
/vsigzip//home/even/my.gz # (absolute path to the .gz)
/vsigzip/c:\users\even\my.gz
```

VSIStatL() will return the uncompressed file size, but this is potentially a slow operation on large files, since it requires uncompressing the whole file. Seeking to the end of the file, or at random locations, is similarly slow. To speed up that process, "snapshots" are internally created in memory so as to be able being able to seek to part of the files already decompressed in a faster way. This mechanism of snapshots also apply to /vsizip/ files.

Write capabilities are also available, but read and write operations cannot be interleaved.

The *GDAL_NUM_THREADS* configuration option can be set to an integer or ALL_CPUS to enable multi-threaded compression of a single file. This is similar to the pigz utility in independent mode. By default the input stream is split into 1 MB chunks (the chunk size can be tuned with the *CPL_VSIL_DEFLATE_CHUNK_SIZE* configuration option, with values like "x K" or "x M"), and each chunk is independently compressed (and terminated by a nine byte marker 0x00 0x00 0xFF 0xFF 0x00 0x00 0x00 0xFF 0xFF, signaling a full flush of the stream and dictionary, enabling potential independent decoding of each chunk). This slightly reduces the compression rate, so very small chunk sizes should be avoided.

8.12.6 /vsitar/ (.tar, .tgz archives)

/vsitar/ is a file handler that allows on-the-fly reading in regular uncompressed .tar or compressed .tgz or .tar.gz archives, without decompressing them in advance.

To point to a file inside a .tar, .tgz .tar.gz file, the filename must be of the form `/vsitar/path/to/the/file.tar/path/inside/the/tar/file`, where `path/to/the/file.tar` is relative or absolute and `path/inside/the/tar/file` is the relative path to the file inside the archive.

To use the .tar as a directory, you can use `/vsizip/path/to/the/file.tar` or `/vsitar/path/to/the/file.tar/subdir`. Directory listing is available with *VSIReadDir()*. A *VSIStatL()* ("/vsitar/...") call will return the uncompressed size of the file. Directories inside the TAR file can be distinguished from regular files with the VSI_ISDIR(stat.st_mode) macro as for regular file systems. Getting directory listing and file statistics are fast operations.

Note: in the particular case where the .tar file contains a single file located at its root, just mentioning `/vsitar/path/to/the/file.tar` will work.

Examples:

```
/vsitar/my.tar/my.tif # (relative path to the .tar)
/vsitar//home/even/my.tar/subdir/my.tif # (absolute path to the .tar)
/vsitar/c:\users\even\my.tar\subdir\my.tif
```

An alternate syntax is available so as to enable chaining and not being dependent on .tar extension, e.g.: `/vsitar/{/path/to/the/archive}/path/inside/the/tar/file`. Note that `/path/to/the/archive` may also itself use this alternate syntax.

8.12.7 /vsi7z/ (.7z archives)

Added in version 3.7.

/vsi7z/ is a file handler that allows reading 7z archives on-the-fly without decompressing them beforehand. This file system is read-only. Directory listing and *VSIStatL()* are available, similarly to above mentioned file systems.

It requires GDAL to be built against libarchive (and libarchive having LZMA support to be of practical use).

To point to a file inside a 7z file, the filename must be of the form `/vsi7z/path/to/the/file.7z/path/inside/the/7z/file`, where `path/to/the/file.7z` is relative or absolute and `path/inside/the/7z/file` is the relative path to the file inside the archive.`

Default extensions recognized by this virtual file system are: `7z`, `lpk` (Esri ArcGIS Layer Package), `lpkx`, `mpk` (Esri ArcGIS Map Package), `mpkx` and `ppkx` (Esri ArcGIS Pro Project Package).

An alternate syntax is available so as to enable chaining and not being dependent on those extensions, e.g.: `/vsi7z/{/path/to/the/archive}/path/inside/the/archive`. Note that `/path/to/the/archive` may also itself use this alternate syntax.

Note that random seeking within a large compressed file will be inefficient when backward seeking is needed (decompression will be restarted from the start of the file). Performance will be the best in sequential reading.

8.12.8 /vsirar/ (.rar archives)

Added in version 3.7.

/vsirar/ is a file handler that allows reading RAR archives on-the-fly without decompressing them beforehand. This file system is read-only. Directory listing and *`VSIStatL()`* are available, similarly to above mentioned file systems.

It requires GDAL to be built against libarchive (and libarchive having LZMA support to be of practical use).

To point to a file inside a RAR file, the filename must be of the form `/vsirar/path/to/the/file.rar/path/inside/the/rar/file`, where `path/to/the/file.rar` is relative or absolute and `path/inside/the/rar/file` is the relative path to the file inside the archive.`

The default extension recognized by this virtual file system is: `rar`

An alternate syntax is available so as to enable chaining and not being dependent on those extensions, e.g.: `/vsirar/{/path/to/the/archive}/path/inside/the/archive`. Note that `/path/to/the/archive` may also itself use this alternate syntax.

Note that random seeking within a large compressed file will be inefficient when backward seeking is needed (decompression will be restarted from the start of the file). Performance will be the best in sequential reading.

8.12.9 Network based file systems

A generic */vsicurl/* file system handler exists for online resources that do not require particular signed authentication schemes. It is specialized into sub-filesystems for commercial cloud storage services, such as */vsis3/*, */vsigs/*, */vsiaz/*, */vsioss/* or */vsiswift/*.

When reading of entire files in a streaming way is possible, prefer using the */vsicurl_streaming/*, and its variants for the above cloud storage services, for more efficiency.

8.12.9.1 /vsicurl/ vs /vsicurl_streaming/ vs HTTP pseudo-driver

After 35 years of existence, one might reasonably hope that reliable access to remote files via the HTTP protocol would be a fully solved problem. While this is mostly true for browser-based experience, the way GDAL accesses data often pushes servers into less-tested territory, requiring users to understand specific low level details.

Below are the options available for "standard" HTTP access (excluding the use of specific file systems such as `/vsigs/` to deal with the specifics of each cloud provider):

- `/vsicurl/`: This virtual file system relies heavily on the server's ability to efficiently handle *HTTP range requests <https://en.wikipedia.org/wiki/Byte_serving>*. A range request includes a header from the client (GDAL) asking the server to return only a specific portion of the resource, defined by start and end offsets. This allows most GDAL drivers to consume datasets piece-wise without downloading the entire file. This is most effective for static resources, such as pre-computed files, rather than content generated on-the-fly for each request. The primary

differences compared to local file access are network latency and bandwidth limits, which `/vsicurl/` attempts to mitigate through in-memory caching and by requesting chunks large enough to justify the request overhead.

Example 1: Getting metadata from a very large remote Cloud Optimized GeoTIFF

```
$ gdal raster info /vsicurl/https://example.com/very_large_cog.tif
```

- `/vsicurl_streaming/`: This virtual file system initiates a background request to download the entire file into memory. It is best suited for drivers that read a dataset sequentially from start to finish with no random seeking. Notably, `/vsicurl_streaming/` does not use HTTP range requests. Of the three options listed here, this is generally the least used. It is particularly ill-suited when reading remote ZIP'ed files, which require reading the end of the file first to get the list of files within the ZIP.

Example 2: Converting a very large remote CSV file to GeoPackage

```
$ gdal vector convert /vsicurl_streaming/https://example.com/very_large.csv out.gpkg
```

Example 3: Copying a remote file to local current directory

```
$ gdal vsi copy /vsicurl_streaming/https://example.com/file.bin .
```

- the `HTTP` (pseudo-driver): This driver downloads the full content of a file into memory before passing it to other GDAL drivers. For modest file sizes that fit easily into RAM and require full processing, this is often the fastest option. It involves a single standard HTTP GET request and does not require range request support. This is frequently the only viable approach for dynamically generated content (often identifiable by the presence of query parameters in the URL). The HTTP pseudo-driver is automatically selected when providing a URL without any `/vsi` prefix.

Example 4: Reprojecting a medium size (fitting into RAM) remote GeoTIFF file

```
$ gdal raster reproject --dst-crs EPSG:32631 -r cubic https://example.com/medium_
↪size.tif out.tif
```

While the HTTP protocol technically allows servers to advertise whether they support range requests (via the Accept-Ranges header), this is not consistently (or efficiently) implemented enough for GDAL to use it as a reliable, automated check for selecting the best access strategy.

8.12.9.2 How to set credentials ?

Cloud storage services require setting credentials. For some of them, they can be provided through configuration files (~/.aws/config, ~/.boto, ..) or through environment variables / configuration options.

Starting with GDAL 3.6, *`VSISetPathSpecificOption()`* can be used to set configuration options with a granularity at the level of a file path, which makes it easier if using the same virtual file system but with different credentials (e.g. different credentials for bucket "/vsis3/foo" and "/vsis3/bar")

Starting with GDAL 3.5, credentials (or path specific options) can be specified in a *GDAL configuration file*, either in a specific one explicitly loaded with *`CPLLoadConfigOptionsFromFile()`*, or one of the default automatically loaded by *`CPLLoadConfigOptionsFromPredefinedFiles()`*.

They should be put under a `[credentials]` section, and for each path prefix, under a relative subsection whose name starts with `[.` (e.g. `[.some_arbitrary_name]`), and whose first key is `path`. ` .. code-block:

```
[credentials]

[.private_bucket]
path=/vsis3/my_private_bucket
AWS_SECRET_ACCESS_KEY=...
AWS_ACCESS_KEY_ID=...

[.sentinel_s2_l1c]
path=/vsis3/sentinel-s2-l1c
AWS_REQUEST_PAYER=requester
```

8.12.9.3 Network/cloud-friendliness and file formats

While most GDAL raster and vector file systems can be accessed in a remote way with /vsicurl/ and other derived virtual file systems, performance is highly dependent on the format, and even for a given format on the special data arrangement. Performance also depends on the particular access pattern made to the file.

For interactive visualisation of raster files, the file should ideally have the following characteristics:

- it should be tiled in generally square-shaped tiles.
- it should have an index of the tile location within the file
- it should have overviews/pyramids

TIFF/GeoTIFF

Cloud-optimized GeoTIFF files as generated by the *COG -- Cloud Optimized GeoTIFF generator* driver are suitable for network access. More generally tiled GeoTIFF files with overviews are.

JPEG2000

JPEG2000 is generally not suitable for network access, unless using a layout carefully designed for that purpose, and when using a JPEG200 library that is heavily optimized.

JPEG2000 files can come in many flavors : single-tiled vs tiled, with different progression order (this is of particular importance for single-tiled access), and with optional markers

The OpenJPEG library (usable through the *JP2OpenJPEG -- JPEG2000 driver based on OpenJPEG library* driver), at the time of writing, needs to ingest each tile-part that participates to the area of interest of the pixel query in a whole (and thus for a single-tiled file, to ingest the whole file). It also does not make use of the potentially present TLM (Tile-Part length) marker, which is the equivalent of a tile index, nor PLT (Packed Length, tile-part header), which is an index of packets within a tile. The Kakadu library (usable through the *JP2KAK -- JPEG 2000 (based on Kakadu SDK)* driver), can use those markers to limit the number of bytes to ingest (but for single-tiled raster, performance might still suffer.)

The dump_jp2.py Python script can be used to check the characteristics of a given JPEG200 file. Fields of interest to examine in the output are:

- the tile size (given by the `XTsiz` and `YTsiz` fields in the `SIZ` marker)
- the presence of `TLM` markers
- the presence of `PLT` markers

8.12.9.4 /vsicurl/ (http/https/ftp files: random access)

/vsicurl/ is a file system handler that allows on-the-fly random reading of files available through HTTP/FTP web protocols, without prior download of the entire file. It requires GDAL to be built against libcurl.

Recognized filenames are of the form `/vsicurl/http[s]://path/to/remote/resource` or `/vsicurl/ftp://path/to/remote/resource`, where `path/to/remote/resource` is the URL of a remote resource.

Example using **ogrinfo** to read a shapefile on the internet:

```
ogrinfo -ro -al -so /vsicurl/https://raw.githubusercontent.com/OSGeo/gdal/master/
↪autotest/ogr/data/poly.shp
```

Options can be passed in the filename with the following syntax: `/vsicurl?[option_i=val_i&]*url=http://...` where each option name and value (including the value of "url") is URL-encoded. Currently supported options are:

- use_head=yes/no: whether the HTTP HEAD request can be emitted. Default to YES. Setting this option overrides the behavior of the *CPL_VSIL_CURL_USE_HEAD* configuration option.
- max_retry=number: default to 0. Setting this option overrides the behavior of the *GDAL_HTTP_MAX_RETRY* configuration option.
- retry_delay=number_in_seconds: default to 30. Setting this option overrides the behavior of the *GDAL_HTTP_RETRY_DELAY* configuration option.
- retry_codes=``ALL`` or comma-separated list of HTTP error codes. Setting this option overrides the behavior of the *GDAL_HTTP_RETRY_CODES* configuration option. (GDAL >= 3.10)
- list_dir=yes/no: whether an attempt to read the file list of the directory where the file is located should be done. Default to YES.
- empty_dir=yes/no: whether to disable directory listing and disable logic in drivers to probe for individual side-car files. Default to NO.
- useragent=value: HTTP UserAgent header
- referer=value: HTTP Referer header
- cookie=value: HTTP Cookie header
- header_file=value: Filename that contains one or several "Header: Value" lines
- header.<key>=<value>: HTTP request header of name <key> and value <value>. (GDAL >= 3.11). e.g. `header.Accept=application%2Fjson`
- unsafessl=yes/no
- low_speed_time=value
- low_speed_limit=value
- proxy=value
- proxyauth=value
- proxyuserpwd=value
- pc_url_signing=yes/no: whether to use the URL signing mechanism of Microsoft Planetary Computer (https://planetarycomputer.microsoft.com/docs/concepts/sas/). (GDAL >= 3.5.2). Note that starting with GDAL 3.9, this may also be set with the path-specific option (cf *VSISetPathSpecificOption()*) `VSICURL_PC_URL_SIGNING` set to `YES`.
- pc_collection=name: name of the collection of the dataset for Planetary Computer URL signing. Only used when pc_url_signing=yes. (GDAL >= 3.5.2)

Partial downloads (requires the HTTP server to support random reading) are done with a 16 KB granularity by default. The chunk size can be configured with the *CPL_VSIL_CURL_CHUNK_SIZE* configuration option, with a value in bytes. If the driver detects sequential reading, it will progressively increase the chunk size up to 128 times *CPL_VSIL_CURL_CHUNK_SIZE* (so 2 MB by default) to improve download performance.

In addition, a global least-recently-used cache of 16 MB shared among all downloaded content is used, and content in it may be reused after a file handle has been closed and reopen, during the life-time of the process or until *VSICurlClearCache()* is called. The size of this global LRU cache can be modified by setting the configuration option *CPL_VSIL_CURL_CACHE_SIZE* (in bytes).

When increasing the value of *CPL_VSIL_CURL_CHUNK_SIZE* to optimize sequential reading, it is recommended to increase *CPL_VSIL_CURL_CACHE_SIZE* as well to 128 times the value of *CPL_VSIL_CURL_CHUNK_SIZE*.

The *GDAL_INGESTED_BYTES_AT_OPEN* configuration option can be set to impose the number of bytes read in one GET call at file opening (can help performance to read Cloud optimized geotiff with a large header).

The *GDAL_HTTP_PROXY* (for both HTTP and HTTPS protocols), *GDAL_HTTPS_PROXY* (for HTTPS protocol only), *GDAL_HTTP_PROXYUSERPWD* and *GDAL_PROXY_AUTH* configuration options can be used to define a proxy server. The syntax to use is the one of Curl `CURLOPT_PROXY`, `CURLOPT_PROXYUSERPWD` and `CURLOPT_PROXYAUTH` options.

The *CURL_CA_BUNDLE* or *SSL_CERT_FILE* configuration options can be used to set the path to the Certification Authority (CA) bundle file (if not specified, curl will use a file in a system location).

Additional HTTP headers can be sent by setting the *GDAL_HTTP_HEADER_FILE* configuration option to point to a filename of a text file with "key: value" HTTP headers.

As an alternative, starting with GDAL 3.6, the *GDAL_HTTP_HEADERS* configuration option can also be used to specify headers. *CPL_CURL_VERBOSE=YES* allows one to see them and more, when combined with `--debug`.

Starting with GDAL 3.10, the `Authorization` header is no longer automatically forwarded when redirections are followed. That behavior can be configured by setting the *CPL_VSIL_CURL_AUTHORIZATION_HEADER_ALLOWED_IF_REDIRECT* configuration option.

Starting with GDAL 3.11, a query string can be appended to a given /vsicurl/ filename by taking its value from the `VSICURL_QUERY_STRING` path-specific option set with *VSISetPathSpecificOption()*. This can for example be used when managing Shared Access Signatures (SAS) on application side, and not wanting to include the signature as part of the filename propagated through GDAL.

The *GDAL_HTTP_MAX_RETRY* (number of attempts) and *GDAL_HTTP_RETRY_DELAY* (in seconds) configuration option can be set, so that request retries are done in case of HTTP errors 429, 502, 503 or 504.

Starting with GDAL 3.6, the following configuration options control the TCP keep-alive functionality (cf https://daniel.haxx.se/blog/2020/02/10/curl-ootw-keepalive-time/ for a detailed explanation):

- *GDAL_HTTP_TCP_KEEPALIVE* = YES/NO. whether to enable TCP keep-alive. Defaults to NO
- *GDAL_HTTP_TCP_KEEPIDLE* = integer, in seconds. Keep-alive idle time. Defaults to 60. Only taken into account if GDAL_HTTP_TCP_KEEPALIVE=YES.
- *GDAL_HTTP_TCP_KEEPINTVL* = integer, in seconds. Interval time between keep-alive probes. Defaults to 60. Only taken into account if GDAL_HTTP_TCP_KEEPALIVE=YES.

Starting with GDAL 3.7, the following configuration options control support for SSL client certificates:

- *GDAL_HTTP_SSLCERT* = filename. Filename of the the SSL client certificate. Cf https://curl.se/libcurl/c/CURLOPT_SSLCERT.html
- *GDAL_HTTP_SSLCERTTYPE* = string. Format of the SSL certificate: "PEM" or "DER". Cf https://curl.se/libcurl/c/CURLOPT_SSLCERTTYPE.html
- *GDAL_HTTP_SSLKEY* = filename. Private key file for TLS and SSL client certificate. Cf https://curl.se/libcurl/c/CURLOPT_SSLKEY.html

- *GDAL_HTTP_KEYPASSWD* = string. Passphrase to private key. Cf https://curl.se/libcurl/c/CURLOPT_KEYPASSWD.html

More generally options of *CPLHTTPFetch()* available through configuration options are available. Starting with GDAL 3.7, the above configuration options can also be specified as path-specific options with *VSISetPathSpecificOption()*.

Starting with GDAL 3.11, the following configuration options control the number of HTTP connections:

- *GDAL_HTTP_MAX_CACHED_CONNECTIONS* = integer_number. Maximum amount of connections that libcurl may keep alive in its connection cache after use. Cf https://curl.se/libcurl/c/CURLMOPT_MAXCONNECTS.html
- *GDAL_HTTP_MAX_TOTAL_CONNECTIONS* = integer_number. Maximum number of simultaneously open connections in total. Cf https://curl.se/libcurl/c/CURLMOPT_MAX_TOTAL_CONNECTIONS.html

The file can be cached in RAM by setting the configuration option *VSI_CACHE* to `TRUE`. The cache size defaults to 25 MB, but can be modified by setting the configuration option *VSI_CACHE_SIZE* (in bytes). Content in that cache is discarded when the file handle is closed.

The *CPL_VSIL_CURL_NON_CACHED* configuration option can be set to values like `/vsicurl/http://example.com/foo.tif:/vsicurl/http://example.com/some_directory`, so that at file handle closing, all cached content related to the mentioned file(s) is no longer cached. This can help when dealing with resources that can be modified during execution of GDAL related code. Alternatively, *VSICurlClearCache()* can be used.

`/vsicurl/` will try to query directly redirected URLs to Amazon S3 signed URLs during their validity period, so as to minimize round-trips. This behavior can be disabled by setting the configuration option *CPL_VSIL_CURL_USE_S3_REDIRECT* to `NO`.

Starting with GDAL 3.12, the *GDAL_HTTP_PATH_VERBATIM* configuration option can be set to `YES` so that sequences of `/../` or `/./` that may exist in the URL's path part are kept unchanged. Otherwise, by default, they are squashed, according to RFC 3986 section 5.2.4.

VSIStatL() will return the size in st_size member and file nature- file or directory - in st_mode member (the later only reliable with FTP resources for now).

VSIReadDir() should be able to parse the HTML directory listing returned by the most popular web servers, such as Apache and Microsoft IIS.

8.12.9.5 /vsicurl_streaming/ (http/https/ftp files: streaming)

/vsicurl_streaming/ is a file system handler that allows on-the-fly sequential reading of files streamed through HTTP/FTP web protocols, without prior download of the entire file. It requires GDAL to be built against libcurl.

Although this file handler is able seek to random offsets in the file, this will not be efficient. If you need efficient random access and that the server supports range downloading, you should use the */vsicurl/* file system handler instead.

Recognized filenames are of the form `/vsicurl_streaming/http[s]://path/to/remote/resource` or `/vsicurl_streaming/ftp://path/to/remote/resource`, where `path/to/remote/resource` is the URL of a remote resource.

The *GDAL_HTTP_PROXY* (for both HTTP and HTTPS protocols), *GDAL_HTTPS_PROXY* (for HTTPS protocol only), *GDAL_HTTP_PROXYUSERPWD* and *GDAL_PROXY_AUTH* configuration options can be used to define a proxy server. The syntax to use is the one of Curl `CURLOPT_PROXY`, `CURLOPT_PROXYUSERPWD` and `CURLOPT_PROXYAUTH` options.

The *CURL_CA_BUNDLE* or *SSL_CERT_FILE* configuration options can be used to set the path to the Certification Authority (CA) bundle file (if not specified, curl will use a file in a system location).

The file can be cached in RAM by setting the configuration option *VSI_CACHE* to `TRUE`. The cache size defaults to 25 MB, but can be modified by setting the configuration option *VSI_CACHE_SIZE* (in bytes).

VSIStatL() will return the size in st_size member and file nature- file or directory - in st_mode member (the later only reliable with FTP resources for now).

8.12.9.6 /vsis3/ (AWS S3 files)

/vsis3/ is a file system handler that allows on-the-fly random reading of (primarily non-public) files available in AWS S3 buckets, without prior download of the entire file. It requires GDAL to be built against libcurl.

It also allows sequential writing of files. No seeks or read operations are then allowed, so in particular direct writing of GeoTIFF files with the GTiff driver is not supported, unless, if, starting with GDAL 3.2, the *CPL_VSIL_USE_TEMP_FILE_FOR_RANDOM_WRITE* configuration option is set to `YES`, in which case random-write access is possible (involves the creation of a temporary local file, whose location is controlled by the *CPL_TMPDIR* configuration option). Deletion of files with *VSIUnlink()* is also supported. Creation of directories with *VSIMkdir()* and deletion of (empty) directories with *VSIRmdir()* are also possible.

Recognized filenames are of the form `/vsis3/bucket/key`, where `bucket` is the name of the S3 bucket and `key` is the S3 object "key", i.e. a filename potentially containing subdirectories.

The generalities of */vsicurl/* apply.

The following configuration options are specific to the /vsis3/ handler:

- **AWS_NO_SIGN_REQUEST=[YES/NO]:** Determines whether to disable request signing.
- **AWS_ACCESS_KEY_ID=value:** Access key ID used for authentication. If using temporary credentials, *AWS_SESSION_TOKEN* must be set.
- **AWS_SECRET_ACCESS_KEY=value:** Secret access key associated with *AWS_ACCESS_KEY_ID*.
- **AWS_SESSION_TOKEN=value:** Session token used for validation of temporary credentials (*AWS_ACCESS_KEY_ID* and *AWS_SECRET_ACCESS_KEY*)
- **AWS_S3SESSION_TOKEN=value:** (GDAL >= 3.12) S3 Express session token (for directory buckets). Can be used by applications handling themselves the CreateSession S3 Express request, in which case they must supply the temporary credentials with *AWS_ACCESS_KEY_ID* and *AWS_SECRET_ACCESS_KEY*
- **CPL_AWS_CREDENTIALS_FILE=<filename>:** Location of an AWS credentials file. If not specified, the standard location of `~/.aws/credentials` will be checked.
- **AWS_DEFAULT_PROFILE=value:** Defaults to `default`. Name of AWS profile.
- **AWS_PROFILE=value:** (GDAL >= 3.2) Defaults to `default`. Name of AWS profile.
- **AWS_CONFIG_FILE=value:** Location of a config file that may provide credentials and the AWS region. if not specified the standard location of `~/.aws/credentials` will be checked.
- **AWS_ROLE_ARN=value:** (GDAL >= 3.6) Amazon Resource Name (ARN) specifying the role to use for authentication via the AssumeRoleWithWebIdentity API.
- **AWS_WEB_IDENTITY_TOKEN_FILE=<filename>:** (GDAL >= 3.6) Duplicate explicit target name: "assumerolewithwebidentity api".

  Path to file with identity token for use for authentication via the AssumeRoleWithWebIdentity API.
- **AWS_REGION=value:** Defaults to `us-east-1`. Set the AWS region to which requests should be sent. Overridden by *AWS_DEFAULT_REGION*.
- **AWS_DEFAULT_REGION=value:** Set the AWS region to which requests should be sent.
- **AWS_REQUEST_PAYER=requester:** Set to `requester` to access a Requester Pays bucket and acknowledge associated charges.
- **AWS_S3_ENDPOINT=value:** Defaults to `s3.amazonaws.com`. Allows the use of /vsis3/ with non-AWS remote object stores that use the AWS S3 protocol. Starting with GDAL 3.11, this can be a URL starting with `http://` or `https://`.

- **AWS_HTTPS=[YES/NO]:** Defaults to YES. If YES, AWS resources will be accessed using HTTPS. If NO, HTTP will be used. No longer needed starting with GDAL 3.11, because *AWS_S3_ENDPOINT* can include the protocol, and when doing so, this option is ignored.
- **AWS_VIRTUAL_HOSTING=[TRUE/FALSE]:** Defaults to TRUE. Select the method of accessing a bucket. If TRUE, identifies the bucket via a virtual bucket host name, e.g.: mybucket.cname.domain.com. If FALSE, identifies the bucket as the top-level directory in the URI, e.g.: cname.domain.com/mybucket
- **VSIS3_CHUNK_SIZE=<MB>:** Defaults to 50. Set the chunk size for multipart uploads.
- **CPL_VSIL_CURL_IGNORE_GLACIER_STORAGE=[YES/NO]:** Defaults to YES. When listing a directory, ignore files with GLACIER storage class. Superseded by *CPL_VSIL_CURL_IGNORE_STORAGE_CLASSES*.
- **CPL_VSIL_CURL_IGNORE_STORAGE_CLASSES=value:** Defaults to GLACIER\,DEEP_ARCHIVE. Comma-separated list of storage class names that should be ignored when listing a directory. If set to empty, objects of all storage classes are retrieved).
- **CPL_VSIS3_USE_BASE_RMDIR_RECURSIVE=[YES/NO]:** (GDAL >= 3.2) Defaults to NO. If YES, recursively delete objects to avoid using batch deletion.
- **CPL_VSIS3_CREATE_DIR_OBJECT=[YES/NO]:** Defaults to YES. Determines whether to allow *VSIMkdir()* to create an empty object to model an empty directory.

Several authentication methods are possible, and are attempted in the following order:

1. If *AWS_NO_SIGN_REQUEST=YES* configuration option is set, request signing is disabled. This option might be used for buckets with public access rights.
2. The *AWS_SECRET_ACCESS_KEY* and *AWS_ACCESS_KEY_ID* configuration options can be set. The *AWS_SESSION_TOKEN* configuration option must be set when temporary credentials are used.
3. Alternate ways of providing credentials similar to what the "aws" command line utility or Boto3 support can be used. If the above mentioned environment variables are not provided, the ~/.aws/credentials or %UserProfile%/.aws/credentials file will be read (or the file pointed by *CPL_AWS_CREDENTIALS_FILE*). The profile may be specified with the *AWS_DEFAULT_PROFILE* environment variable, or starting with GDAL 3.2 with the *AWS_PROFILE* environment variable (the default profile is "default").
4. The ~/.aws/config or %UserProfile%/.aws/config file may also be used (or the file pointer by *AWS_CONFIG_FILE*) to retrieve credentials and the AWS region.
5. Starting with GDAL 3.6, if *AWS_ROLE_ARN* and *AWS_WEB_IDENTITY_TOKEN_FILE* are defined we will rely on credentials mechanism for web identity token based AWS STS action AssumeRoleWithWebIdentity (See.: https://docs.aws.amazon.com/eks/latest/userguide/iam-roles-for-service-accounts.html)
6. If none of the above method succeeds, instance profile credentials will be retrieved when GDAL is used on EC2 instances (cf */vsis3/ and AWS Instance Metadata Service (IMDS)*)

On writing, the file is uploaded using the S3 multipart upload API. The size of chunks is set to 50 MB by default, allowing creating files up to 500 GB (10000 parts of 50 MB each). If larger files are needed, then increase the value of the *VSIS3_CHUNK_SIZE* config option to a larger value (expressed in MB). In case the process is killed and the file not properly closed, the multipart upload will remain open, causing Amazon to charge you for the parts storage. You'll have to abort yourself with other means such "ghost" uploads (e.g. with the s3cmd utility) For files smaller than the chunk size, a simple PUT request is used instead of the multipart upload API.

Since GDAL 3.1, the *VSIRename()* operation is supported (first doing a copy of the original file and then deleting it)

Since GDAL 3.1, the *VSIRmdirRecursive()* operation is supported (using batch deletion method). The *CPL_VSIS3_USE_BASE_RMDIR_RECURSIVE* configuration option can be set to YES if using a S3-like API that doesn't support batch deletion (GDAL >= 3.2). Starting with GDAL 3.6, this can be set as a path-specific option in the *GDAL configuration file*

The *CPL_VSIS3_CREATE_DIR_OBJECT* configuration option can be set to NO to prevent the *VSIMkdir()* operation from creating an empty object with the name of the directory terminated with a slash directory. By default GDAL creates such object, so that empty directories can be modeled, but this may cause compatibility problems with applications that do not expect such empty objects.

Starting with GDAL 3.5, profiles that use IAM role assumption (see https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-role.html) are handled. The `role_arn` and `source_profile` keywords are required in such profiles. The optional `external_id`, `mfa_serial` and `role_session_name` can be specified. `credential_source` is not supported currently.

Support for AWS Single-Sign On (AWS IAM Identity Center) parameters in `~/.aws/config` and cached SSO files in `~/.aws/sso/cache` is implemented since GDAL 3.10.1.

8.12.9.7 /vsis3/ and AWS Instance Metadata Service (IMDS)

On EC2 instances, GDAL will try to use the IMDSv2 protocol in priority to get the authentication tokens for AWS S3, and fallback to IMDSv1 in case of failure. Note however that on recent Amazon Linux instances, IMDSv1 is no longer accessible, and thus IMDSv2 must be correctly configured (and even if IMDSv1 is available, mis-configured IMDSv2 will cause delays in the authentication step).

There are known issues when running inside a Docker instance in a EC2 instance that require extra configuration of the instance. For example, you need to increase the hop limit to 2

There are several ways to do this. One way is to run this command:

```
aws ec2 modify-instance-metadata-options \
    --instance-id <instance_id> \
    --http-put-response-hop-limit 2 \
    --http-endpoint enabled
```

Another is to set the HttpPutResponseHopLimit metadata on an AutoScalingGroup LaunchTemplate: - https://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_InstanceMetadataOptionsRequest.html - https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-launchtemplate-metadataoptions.html

Another possibility is to start the Docker container with host networking (`--network=host`), although this breaks isolation of containers by exposing all ports of the host to the container and has thus security implications.

8.12.9.8 Configuring /vsis3/ with Minio

The following configuration options can be set to access a Minio Docker image

- AWS_VIRTUAL_HOSTING=FALSE
- AWS_HTTPS=NO
- AWS_S3_ENDPOINT="localhost:9000"
- AWS_REGION="us-east-1"
- AWS_SECRET_ACCESS_KEY="your_secret_access_key"
- AWS_ACCESS_KEY_ID="your_access_key"

8.12.9.9 /vsis3_streaming/ (AWS S3 files: streaming)

/vsis3_streaming/ is a file system handler that allows on-the-fly sequential reading of (primarily non-public) files available in AWS S3 buckets, without prior download of the entire file. It requires GDAL to be built against libcurl.

Recognized filenames are of the form `/vsis3_streaming/bucket/key` where `bucket` is the name of the S3 bucket and `key` is the S3 object "key", i.e. a filename potentially containing subdirectories.

Authentication options, and read-only features, are identical to */vsis3/*

8.12.9.10 /vsigs/ (Google Cloud Storage files)

/vsigs/ is a file system handler that allows on-the-fly random reading of (primarily non-public) files available in Google Cloud Storage buckets, without prior download of the entire file. It requires GDAL to be built against libcurl.

It also allows sequential writing of files. No seeks or read operations are then allowed, so in particular direct writing of GeoTIFF files with the GTiff driver is not supported, unless, if, starting with GDAL 3.2, the *CPL_VSIL_USE_TEMP_FILE_FOR_RANDOM_WRITE* configuration option is set to `YES`, in which case random-write access is possible (involves the creation of a temporary local file, whose location is controlled by the *CPL_TMPDIR* configuration option). Deletion of files with *VSIUnlink()*, creation of directories with *VSIMkdir()* and deletion of (empty) directories with *VSIRmdir()* are also possible.

Recognized filenames are of the form `/vsigs/bucket/key` where `bucket` is the name of the bucket and `key` is the object "key", i.e. a filename potentially containing subdirectories.

The generalities of */vsicurl/* apply.

The following configuration options are specific to the /vsigs/ handler:

- **GS_NO_SIGN_REQUEST=[YES/NO]:** (GDAL >= 3.4) If `YES`, request signing is disabled.
- **GS_SECRET_ACCESS_KEY=value:** Secret for AWS-style authentication (HMAC keys).
- **GS_ACCESS_KEY_ID=value:** Access ID for AWS-style authentication (HMAC keys).
- **GS_OAUTH2_REFRESH_TOKEN=value:** OAuth2 refresh token. This refresh token can be obtained with the gdal_auth.py script (`gdal_auth.py -s storage` or `gdal_auth.py -s storage-rw`).
- **GS_OAUTH2_CLIENT_ID=value:** Client ID to be used when requesting *GS_OAUTH2_REFRESH_TOKEN*.
- **GS_OAUTH2_CLIENT_SECRET=value:** Client secret to be used when requesting *GS_OAUTH2_REFRESH_TOKEN*.
-
- **GS_OAUTH2_PRIVATE_KEY=value:** Private key for OAuth2 authentication. Alternatively, the key may be saved in a file and referenced with *GS_OAUTH2_PRIVATE_KEY_FILE*.
- **GS_OAUTH2_PRIVATE_KEY_FILE=<filename>:** Location of private key file for OAuth2 authentication.
- **GS_OAUTH2_CLIENT_EMAIL=value:** Client email for OAuth2 authentication, to be used with *GS_OAUTH2_PRIVATE_KEY* or *GS_OAUTH2_PRIVATE_KEY_FILE*.
- **GS_OAUTH2_SCOPE=value:** Permission scope associated with OAuth2 authentication using *GOOGLE_APPLICATION_CREDENTIALS*.
- **CPL_GS_CREDENTIALS_FILE=value:** Defaults to `~/.boto`. Location of configuration file providing `gs_secret_access_key` and `gs_access_key_id`.
- **GS_USER_PROJECT=value:** (GDAL >= 3.4) Google Project id (see https://cloud.google.com/storage/docs/xml-api/reference-headers#xgooguserproject) to charge for requests against Requester Pays buckets.

Several authentication methods are possible, and are attempted in the following order:

1. If *GS_NO_SIGN_REQUEST=YES* configuration option is set, request signing is disabled. This option might be used for buckets with public access rights. Available since GDAL 3.4
2. The *GS_SECRET_ACCESS_KEY* and *GS_ACCESS_KEY_ID* configuration options can be set for AWS-style authentication
3. The *GDAL_HTTP_HEADER_FILE* configuration option to point to a filename of a text file with "key: value" headers. Typically, it must contain a "Authorization: Bearer XXXXXXXXX" line.
4. (GDAL >= 3.7) The *GDAL_HTTP_HEADERS* configuration option can also be set. It must contain at least a line starting with "Authorization:" to be used as an authentication method.

5. The *GS_OAUTH2_REFRESH_TOKEN* configuration option can be set to use OAuth2 client authentication. See http://code.google.com/apis/accounts/docs/OAuth2.html This refresh token can be obtained with the `gdal_auth.py -s storage` or `gdal_auth.py -s storage-rw` script Note: instead of using the default GDAL application credentials, you may define the *GS_OAUTH2_CLIENT_ID* and *GS_OAUTH2_CLIENT_SECRET* configuration options (need to be defined both for gdal_auth.py and later execution of /vsigs)
6. The *GOOGLE_APPLICATION_CREDENTIALS* configuration option can be set to point to a JSON file containing OAuth2 service account credentials (`type: service_account`), in particular a private key and a client email. See https://developers.google.com/identity/protocols/OAuth2ServiceAccount for more details on this authentication method. The bucket must grant the "Storage Legacy Bucket Owner" or "Storage Legacy Bucket Reader" permissions to the service account. The *GS_OAUTH2_SCOPE* configuration option can be set to change the default permission scope from "https://www.googleapis.com/auth/devstorage.read_write" to "https://www.googleapis.com/auth/devstorage.read_only" if needed.
7. (GDAL >= 3.4.2) The *GOOGLE_APPLICATION_CREDENTIALS* configuration option can be set to point to a JSON file containing OAuth2 user credentials (`type: authorized_user`).
8. Variant of the previous method. The *GS_OAUTH2_PRIVATE_KEY* (or *GS_OAUTH2_PRIVATE_KEY_FILE* and *GS_OAUTH2_CLIENT_EMAIL* can be set to use OAuth2 service account authentication. See https://developers.google.com/identity/protocols/OAuth2ServiceAccount for more details on this authentication method. The *GS_OAUTH2_PRIVATE_KEY* configuration option must contain the private key as a inline string, starting with `-----BEGIN PRIVATE KEY-----`. Alternatively the *GS_OAUTH2_PRIVATE_KEY_FILE* configuration option can be set to indicate a filename that contains such a private key. The bucket must grant the "Storage Legacy Bucket Owner" or "Storage Legacy Bucket Reader" permissions to the service account. The *GS_OAUTH2_SCOPE* configuration option can be set to change the default permission scope from "https://www.googleapis.com/auth/devstorage.read_write" to "https://www.googleapis.com/auth/devstorage.read_only" if needed.
9. An alternate way of providing credentials similar to what the "gsutil" command line utility or Boto3 support can be used. If the above mentioned environment variables are not provided, the `~/.boto` or `UserProfile%/.boto` file will be read (or the file pointed by *CPL_GS_CREDENTIALS_FILE*) for the gs_secret_access_key and gs_access_key_id entries for AWS style authentication. If not found, it will look for the gs_oauth2_refresh_token (and optionally client_id and client_secret) entry for OAuth2 client authentication.
10. Finally if none of the above method succeeds, the code will check if the current machine is a Google Compute Engine instance, and if so will use the permissions associated to it (using the default service account associated with the VM). To force a machine to be detected as a GCE instance (for example for code running in a container with no access to the boot logs), you can set *CPL_MACHINE_IS_GCE* to `YES`.

Since GDAL 3.1, the Rename() operation is supported (first doing a copy of the original file and then deleting it).

8.12.9.11 /vsigs_streaming/ (Google Cloud Storage files: streaming)

/vsigs_streaming/ is a file system handler that allows on-the-fly sequential reading of files (primarily non-public) files available in Google Cloud Storage buckets, without prior download of the entire file. It requires GDAL to be built against libcurl.

Recognized filenames are of the form `/vsigs_streaming/bucket/key` where `bucket` is the name of the bucket and `key` is the object "key", i.e. a filename potentially containing subdirectories.

Authentication options, and read-only features, are identical to */vsigs/*

8.12.9.12 /vsiaz/ (Microsoft Azure Blob files)

/vsiaz/ is a file system handler that allows on-the-fly random reading of (primarily non-public) files available in Microsoft Azure Blob containers, without prior download of the entire file. It requires GDAL to be built against libcurl.

See */vsiadls/* for a related filesystem for Azure Data Lake Storage Gen2.

It also allows sequential writing of files. No seeks or read operations are then allowed, so in particular direct writing of GeoTIFF files with the GTiff driver is not supported, unless, if, starting with GDAL 3.2, the

CPL_VSIL_USE_TEMP_FILE_FOR_RANDOM_WRITE configuration option is set to YES, in which case random-write access is possible (involves the creation of a temporary local file, whose location is controlled by the *CPL_TMPDIR* configuration option). A block blob will be created if the file size is below 4 MB. Beyond, an append blob will be created (with a maximum file size of 195 GB).

Deletion of files with *VSIUnlink()*, creation of directories with *VSIMkdir()* and deletion of (empty) directories with *VSIRmdir()* are also possible. Note: when using *VSIMkdir()*, a special hidden `.gdal_marker_for_dir` empty file is created, since Azure Blob does not natively support empty directories. If that file is the last one remaining in a directory, *VSIRmdir()* will automatically remove it. This file will not be seen with *VSIReadDir()*. If removing files from directories not created with *VSIMkdir()*, when the last file is deleted, its directory is automatically removed by Azure, so the sequence `VSIUnlink("/vsiaz/container/subdir/lastfile")` followed by `VSIRmdir("/vsiaz/container/subdir")` will fail on the *VSIRmdir()* invocation.

Recognized filenames are of the form `/vsiaz/container/key`, where `container` is the name of the container and `key` is the object "key", i.e. a filename potentially containing subdirectories.

The generalities of */vsicurl/* apply.

The following configuration options are specific to the /vsiaz/ handler:

- **AZURE_NO_SIGN_REQUEST=[YES/NO]:** (GDAL >= 3.2) Controls whether requests are signed.
- **AZURE_STORAGE_CONNECTION_STRING=value:** Credential string provided in the Access Key section of the administrative interface, containing both the account name and a secret key.
- **AZURE_STORAGE_ACCESS_TOKEN=value:** (GDAL >= 3.5) Access token typically obtained using Microsoft Authentication Library (MSAL).
- **AZURE_STORAGE_ACCOUNT=value:** Specifies storage account name.
- **AZURE_STORAGE_ACCESS_KEY=value:** Specifies secret key associated with *AZURE_STORAGE_ACCOUNT*.
- **AZURE_STORAGE_SAS_TOKEN=value:** (GDAL >= 3.2) Shared Access Signature.
- **AZURE_IMDS_OBJECT_ID=value:** (GDAL >= 3.8) object_id of the managed identity you would like the token for, when using Azure Instance Metadata Service (IMDS) authentication in a Azure Virtual Machine. Required if your VM has multiple user-assigned managed identities. This option may be set as a path-specific option with *VSISetPathSpecificOption()*
- **AZURE_IMDS_CLIENT_ID=value:** (GDAL >= 3.8) client_id of the managed identity you would like the token for, when using Azure Instance Metadata Service (IMDS) authentication in a Azure Virtual Machine. Required if your VM has multiple user-assigned managed identities. This option may be set as a path-specific option with *VSISetPathSpecificOption()*
- **AZURE_IMDS_MSI_RES_ID=value:** (GDAL >= 3.8) msi_res_id (Azure Resource ID) of the managed identity you would like the token for, when using Azure Instance Metadata Service (IMDS) authentication in a Azure Virtual Machine. Required if your VM has multiple user-assigned managed identities. This option may be set as a path-specific option with *VSISetPathSpecificOption()*

Several authentication methods are possible, and are attempted in the following order:

1. The *AZURE_STORAGE_CONNECTION_STRING* configuration option
2. The *AZURE_STORAGE_ACCOUNT* configuration option is set to specify the account name AND
   a) (GDAL >= 3.5) The *AZURE_STORAGE_ACCESS_TOKEN* configuration option is set to specify the access token, that will be included in a "Authorization: Bearer ${AZURE_STORAGE_ACCESS_TOKEN}" header. This access token is typically obtained using Microsoft Authentication Library (MSAL).
   b) The *AZURE_STORAGE_ACCESS_KEY* configuration option is set to specify the secret key.

c) The *AZURE_NO_SIGN_REQUEST=YES* configuration option is set, so as to disable any request signing. This option might be used for accounts with public access rights. Available since GDAL 3.2

d) The *AZURE_STORAGE_SAS_TOKEN* configuration option (AZURE_SAS if GDAL < 3.5) is set to specify a Shared Access Signature. This SAS is appended to URLs built by the /vsiaz/ file system handler. Its value should already be URL-encoded and should not contain any leading '?' or '&' character (e.g. a valid one may look like "st=2019-07-18T03%3A53%3A22Z&se=2035-07-19T03%3A53%3A00Z&sp=rl&sv=2018-03-28&sr=c&sig=2RIXmLbLbiagYnUd49rgx2kOXKyILrJOgafmkODhRAQ%3D"). Available since GDAL 3.2

e) The current machine is a Azure Virtual Machine with Azure Active Directory permissions assigned to it (see https://docs.microsoft.com/en-us/azure/active-directory/managed-identities-azure-resources/qs-configure-portal-windows-vm). Available since GDAL 3.3.

Authentication using Azure Active Directory Workload Identity (using AZURE_TENANT_ID, AZURE_CLIENT_ID, AZURE_FEDERATED_TOKEN_FILE and AZURE_AUTHORITY_HOST environment variables), typically for Azure Kubernetes, is available since GDAL 3.7.2

3. Starting with GDAL 3.5, the *configuration file <https://github.com/MicrosoftDocs/azure-docs-cli/blob/main/docs-ref-conceptual/azure-cli-configuration.md>* of the "az" command line utility can be used. The following keys of the `[storage]` section will be used in the following priority: `connection_string`, `account` + `key` or `account` + `sas_token`

Since GDAL 3.1, the *VSIRename()* operation is supported (first doing a copy of the original file and then deleting it)

Since GDAL 3.3, the *VSIGetFileMetadata()* and *VSISetFileMetadata()* operations are supported.

8.12.9.13 /vsiaz_streaming/ (Microsoft Azure Blob files: streaming)

/vsiaz_streaming/ is a file system handler that allows on-the-fly sequential reading of files (primarily non-public) files available in Microsoft Azure Blob containers, buckets, without prior download of the entire file. It requires GDAL to be built against libcurl.

Recognized filenames are of the form `/vsiaz_streaming/container/key` where `container` is the name of the container and `key` is the object "key", i.e. a filename potentially containing subdirectories.

Authentication options, and read-only features, are identical to */vsiaz/*

8.12.9.14 /vsiadls/ (Microsoft Azure Data Lake Storage Gen2)

/vsiadls/ is a file system handler that allows on-the-fly random reading of (primarily non-public) files available in Microsoft Azure Data Lake Storage file systems, without prior download of the entire file. It requires GDAL to be built against libcurl.

It has similar capabilities as */vsiaz/*, and in particular uses the same configuration options for authentication. Its advantages over /vsiaz/ are a real management of directory and Unix-style ACL support. Some features require the Azure storage to have hierarchical support turned on. Consult its documentation

The main enhancements over /vsiaz/ are:

- True directory support (no need for the artificial `.gdal_marker_for_dir` empty file that is used for /vsiaz/ to have empty directories)
- One-call recursive directory deletion with *VSIRmdirRecursive()*
- Atomic renaming with *VSIRename()*
- *VSIGetFileMetadata()* support for the "STATUS" and "ACL" metadata domains

- *VSISetFileMetadata()* support for the "PROPERTIES" and "ACL" metadata domains

Added in version 3.3.

8.12.9.15 /vsioss/ (Alibaba Cloud OSS files)

/vsioss/ is a file system handler that allows on-the-fly random reading of (primarily non-public) files available in Alibaba Cloud Object Storage Service (OSS) buckets, without prior download of the entire file. It requires GDAL to be built against libcurl.

It also allows sequential writing of files. No seeks or read operations are then allowed, so in particular direct writing of GeoTIFF files with the GTiff driver is not supported, unless, if, starting with GDAL 3.2, the *CPL_VSIL_USE_TEMP_FILE_FOR_RANDOM_WRITE* configuration option is set to `YES`, in which case random-write access is possible (involves the creation of a temporary local file, whose location is controlled by the *CPL_TMPDIR* configuration option). Deletion of files with *VSIUnlink()* is also supported. Creation of directories with *VSIMkdir()* and deletion of (empty) directories with *VSIRmdir()* are also possible.

Recognized filenames are of the form `/vsioss/bucket/key` where `bucket` is the name of the OSS bucket and `key` is the OSS object "key", i.e. a filename potentially containing subdirectories.

The generalities of */vsicurl/* apply.

The following configuration options are specific to the /vsioss/ handler:

- **OSS_SECRET_ACCESS_KEY=value:** (required) Secret access key for authentication.
- **OSS_ACCESS_KEY_ID=value:** (required) Access key ID for authentication.
- **OSS_ENDPOINT=value:** Defaults to `oss-us-east-1.aliyuncs.com`. Endpoint URL containing the region associated with the bucket.
- **VSIOSS_CHUNK_SIZE=<MB>:** Defaults to `50`. Chunk size used with multipart upload API.

The *OSS_SECRET_ACCESS_KEY* and *OSS_ACCESS_KEY_ID* configuration options must be set. The *OSS_ENDPOINT* configuration option should normally be set to the appropriate value, which reflects the region attached to the bucket. If the bucket is stored in another region than oss-us-east-1, the code logic will redirect to the appropriate endpoint.

On writing, the file is uploaded using the OSS multipart upload API. The size of chunks is set to 50 MB by default, allowing creating files up to 500 GB (10000 parts of 50 MB each). If larger files are needed, then increase the value of the *VSIOSS_CHUNK_SIZE* config option to a larger value (expressed in MB). In case the process is killed and the file not properly closed, the multipart upload will remain open, causing Alibaba to charge you for the parts storage. You'll have to abort yourself with other means. For files smaller than the chunk size, a simple PUT request is used instead of the multipart upload API.

8.12.9.16 /vsioss_streaming/ (Alibaba Cloud OSS files: streaming)

/vsioss_streaming/ is a file system handler that allows on-the-fly sequential reading of files (primarily non-public) files available in Alibaba Cloud Object Storage Service (OSS) buckets, without prior download of the entire file. It requires GDAL to be built against libcurl.

Recognized filenames are of the form `/vsioss_streaming/bucket/key` where `bucket` is the name of the bucket and `key` is the object "key", i.e. a filename potentially containing subdirectories.

Authentication options, and read-only features, are identical to */vsioss/*

8.12.9.17 /vsiswift/ (OpenStack Swift Object Storage)

/vsiswift/ is a file system handler that allows on-the-fly random reading of (primarily non-public) files available in OpenStack Swift Object Storage (swift) buckets, without prior download of the entire file. It requires GDAL to be built against libcurl.

It also allows sequential writing of files. No seeks or read operations are then allowed, so in particular direct writing of GeoTIFF files with the GTiff driver is not supported, unless, if, starting with GDAL 3.2, the *CPL_VSIL_USE_TEMP_FILE_FOR_RANDOM_WRITE* configuration option is set to `YES`, in which case random-write access is possible (involves the creation of a temporary local file, whose location is controlled by the *CPL_TMPDIR* configuration option). Deletion of files with *VSIUnlink()* is also supported. Creation of directories with *VSIMkdir()* and deletion of (empty) directories with *VSIRmdir()* are also possible.

Recognized filenames are of the form `/vsiswift/bucket/key` where `bucket` is the name of the swift bucket and `key` is the swift object "key", i.e. a filename potentially containing subdirectories.

The generalities of */vsicurl/* apply.

The following configuration options are specific to the /vsioss/ handler:

- **SWIFT_STORAGE_URL=value:** Storage URL.
- **SWIFT_AUTH_TOKEN=value:** Value of the x-auth-token authorization
- **SWIFT_AUTH_V1_URL=value:** URL for Auth V1 authentication.
- **SWIFT_USER=value:** User name for Auth V1 authentication.
- **SWIFT_KEY=value:** Key for Auth V1 authentication.

Three authentication methods are possible, and are attempted in the following order:

1. The *SWIFT_STORAGE_URL* and *SWIFT_AUTH_TOKEN* configuration options are set respectively to the storage URL (e.g http://127.0.0.1:12345/v1/AUTH_something) and the value of the x-auth-token authorization token.
2. The *SWIFT_AUTH_V1_URL*, *SWIFT_USER* and *SWIFT_KEY* configuration options are set respectively to the endpoint of the Auth V1 authentication (e.g http://127.0.0.1:12345/auth/v1.0), the user name and the key/password. This authentication endpoint will be used to retrieve the storage URL and authorization token mentioned in the first authentication method.
3. Authentication with Keystone v3 is using the same options as python-swiftclient, see https://docs.openstack.org/python-swiftclient/latest/cli/index.html#authentication for more details. GDAL (>= 3.1) supports the following options:
   - *OS_IDENTITY_API_VERSION=3*
   - *OS_AUTH_URL*
   - *OS_USERNAME*
   - *OS_PASSWORD*
   - *OS_USER_DOMAIN_NAME*
   - *OS_PROJECT_NAME*
   - *OS_PROJECT_DOMAIN_NAME*
   - *OS_REGION_NAME*
4. Application Credential Authentication via Keystone v3, GDAL (>= 3.3.1) supports application-credential authentication with the following options:
   - *OS_IDENTITY_API_VERSION=3*
   - *OS_AUTH_TYPE=v3applicationcredential*
   - *OS_AUTH_URL*
   - *OS_APPLICATION_CREDENTIAL_ID*
   - *OS_APPLICATION_CREDENTIAL_SECRET*

- *OS_REGION_NAME*

This file system handler also allows sequential writing of files (no seeks or read operations are then allowed).

In some versions of OpenStack Swift, the access to large (segmented) files fails unless they are explicitly marked as static large objects, instead of being dynamic large objects which is the default. Using the python-swiftclient this can be achieved when uploading the file by passing the `--use-slo` flag (see https://docs.openstack.org/python-swiftclient/latest/cli/index.html#swift-upload for all options). For more information about large objects see https://docs.openstack.org/swift/latest/api/large_objects.html.

8.12.9.18 /vsiswift_streaming/ (OpenStack Swift Object Storage: streaming)

/vsiswift_streaming/ is a file system handler that allows on-the-fly sequential reading of files (primarily non-public) files available in OpenStack Swift Object Storage (swift) buckets, without prior download of the entire file. It requires GDAL to be built against libcurl.

Recognized filenames are of the form `/vsiswift_streaming/bucket/key` where `bucket` is the name of the bucket and `key` is the object "key", i.e. a filename potentially containing subdirectories.

Authentication options, and read-only features, are identical to */vsiswift/*

8.12.9.19 /vsihdfs/ (Hadoop File System)

/vsihdfs/ is a file system handler that provides read access to HDFS. This handler requires GDAL to have been built with Java support (CMake FindJNI) and HDFS support. Support for this handler is currently only available on Unix-like systems.

Note: support for the HTTP REST API (webHdfs) is also available with */vsiwebhdfs/ (Web Hadoop File System REST API)*

The LD_LIBRARY_PATH and CLASSPATH environment variables must be typically set up as following.

```
HADOOP_HOME=$HOME/hadoop-3.3.5
LD_LIBRARY_PATH=$HADOOP_HOME/lib/native:$LD_LIBRARY_PATH
CLASSPATH=$HADOOP_HOME/etc/hadoop:$HADOOP_HOME/share/hadoop/common/*:$HADOOP_HOME/share/
↪hadoop/common/lib/*:$HADOOP_HOME/share/hadoop/hdfs/*
```

Failure to properly define the CLASSPATH will result in hard crashes in the native libhdfs.

Relevant Hadoop documentation links:

- C API libhdfs
- HDFS Users Guide
- Hadoop: Setting up a Single Node Cluster

Recognized filenames are of the form `/vsihdfs/hdfsUri` where `hdfsUri` is a valid HDFS URI.

Examples:

```
/vsihdfs/file:/home/user//my.tif  (a local file accessed through HDFS)
/vsihdfs/hdfs://localhost:9000/my.tif  (a file stored in HDFS)
```

8.12.9.20 /vsiwebhdfs/ (Web Hadoop File System REST API)

/vsiwebhdfs/ is a file system handler that provides read and write access to HDFS through its HTTP REST API.

Recognized filenames are of the form `/vsiwebhdfs/http://hostname:port/webhdfs/v1/path/to/filename`.

Examples:

```
/vsiwebhdfs/http://localhost:50070/webhdfs/v1/mydir/byte.tif
```

It also allows sequential writing of files. No seeks or read operations are then allowed, so in particular direct writing of GeoTIFF files with the GTiff driver is not supported, unless, if, starting with GDAL 3.2, the *CPL_VSIL_USE_TEMP_FILE_FOR_RANDOM_WRITE* configuration option is set to `YES`, in which case random-write access is possible (involves the creation of a temporary local file, whose location is controlled by the *CPL_TMPDIR* configuration option). Deletion of files with *VSIUnlink()* is also supported. Creation of directories with *VSIMkdir()* and deletion of (empty) directories with *VSIRmdir()* are also possible.

The generalities of */vsicurl/* apply.

The following configuration options are available:

- **WEBHDFS_USERNAME=value:** User name (when security is off).
- **WEBHDFS_DELEGATION=value:** Hadoop delegation token (when security is on).
- **WEBHDFS_DATANODE_HOST=value:** For APIs using redirect, substitute the redirection hostname with the one provided by this option (normally resolvable hostname should be rewritten by a proxy)
- **WEBHDFS_REPLICATION=<integer>:** Replication value used when creating a file
- **WEBHDFS_PERMISSION=<integer>:** Permission mask (to provide as decimal number) when creating a file or directory

This file system handler also allows sequential writing of files (no seeks or read operations are then allowed)

8.12.10 /vsistdin/ (standard input streaming)

/vsistdin/ is a file handler that allows reading from the standard input stream.

The filename syntax must be only `/vsistdin/`, (not e.g., /vsistdin/path/to/f.csv , but "/vsistdin?buffer_limit=value" is OK.)

The file operations available are of course limited to Read() and forward Seek(). Full seek in the first MB of a file is possible, and it is cached so that closing, re-opening `/vsistdin/` and reading within this first megabyte is possible multiple times in the same process.

The size of the in-memory cache can be controlled with the *CPL_VSISTDIN_BUFFER_LIMIT* configuration option:

- **CPL_VSISTDIN_BUFFER_LIMIT=value:** (GDAL >= 3.6) Defaults to `1MB`. Specifies the size of the /vsistdin in bytes (or using a MB or GB suffix, e.g. "1GB"), or -1 for unlimited.

The "/vsistdin?buffer_limit=value" syntax can also be used.

/vsistdin filenames can be combined with other file system. For example, to read a file within a potentially big ZIP file streamed to gdal_translate:

```
cat file.tif.zip | gdal_translate /vsizip/{/vsistdin?buffer_limit=-1}/path/to/some.tif␣
↪out.tif
```

8.12.11 /vsistdout/ (standard output streaming)

/vsistdout/ is a file handler that allows writing into the standard output stream.

The filename syntax must be only `/vsistdout/`.

The file operations available are of course limited to Write().

A variation of this file system exists as the `/vsistdout_redirect/` file system handler, where the output function can be defined with *VSIStdoutSetRedirection()*.

8.12.12 /vsimem/ (in-memory files)

/vsimem/ is a file handler that allows block of memory to be treated as files. All portions of the file system underneath the base path `/vsimem/` will be handled by this driver.

Normal VSI*L functions can be used freely to create and destroy memory arrays, treating them as if they were real file system objects. Some additional methods exist to efficiently create memory file system objects without duplicating original copies of the data or to "steal" the block of memory associated with a memory file. See *VSIFileFromMemBuffer()* and *VSIGetMemFileBuffer()*.

Directory related functions are supported.

/vsimem/ files are visible within the same process. Multiple threads can access the same underlying file in read mode, provided they used different handles, but concurrent write and read operations on the same underlying file are not supported (locking is left to the responsibility of calling code).

8.12.13 /vsisubfile/ (portions of files)

The /vsisubfile/ virtual file system handler allows access to subregions of files, treating them as a file on their own to the virtual file system functions (VSIFOpenL(), etc).

A special form of the filename is used to indicate a subportion of another file: `/vsisubfile/<offset>[_<size>],<filename>`.

The size parameter is optional. Without it the remainder of the file from the start offset as treated as part of the subfile. Otherwise only <size> bytes from <offset> are treated as part of the subfile. The <filename> portion may be a relative or absolute path using normal rules. The <offset> and <size> values are in bytes.

Examples:

```
/vsisubfile/1000_3000,/data/abc.ntf
/vsisubfile/5000,../xyz/raw.dat
```

Unlike the /vsimem/ or conventional file system handlers, there is no meaningful support for filesystem operations for creating new files, traversing directories, and deleting files within the /vsisubfile/ area. Only the *VSIStatL()*, *VSIFOpenL()* and operations based on the file handle returned by *VSIFOpenL()* operate properly.

8.12.14 /vsisparse/ (sparse files)

The /vsisparse/ virtual file handler allows a virtual file to be composed from chunks of data in other files, potentially with large spaces in the virtual file set to a constant value. This can make it possible to test some sorts of operations on what seems to be a large file with image data set to a constant value. It is also helpful when wanting to add test files to the test suite that are too large, but for which most of the data can be ignored. It could, in theory, also be used to treat several files on different file systems as one large virtual file.

The file referenced by /vsisparse/ should be an XML control file formatted something like:

```
<VSISparseFile>
    <Length>87629264</Length>
    <SubfileRegion>  <!-- Stuff at start of file. -->
        <Filename relative="1">251_head.dat</Filename>
        <DestinationOffset>0</DestinationOffset>
        <SourceOffset>0</SourceOffset>
        <RegionLength>2768</RegionLength>
    </SubfileRegion>

    <SubfileRegion>  <!-- RasterDMS node. -->
        <Filename relative="1">251_rasterdms.dat</Filename>
```

(continues on next page)

(continued from previous page)

```
        <DestinationOffset>87313104</DestinationOffset>
        <SourceOffset>0</SourceOffset>
        <RegionLength>160</RegionLength>
    </SubfileRegion>

    <SubfileRegion>  <!-- Stuff at end of file. -->
        <Filename relative="1">251_tail.dat</Filename>
        <DestinationOffset>87611924</DestinationOffset>
        <SourceOffset>0</SourceOffset>
        <RegionLength>17340</RegionLength>
    </SubfileRegion>

    <ConstantRegion>  <!-- Default for the rest of the file. -->
        <DestinationOffset>0</DestinationOffset>
        <RegionLength>87629264</RegionLength>
        <Value>0</Value>
    </ConstantRegion>
</VSISparseFile>
```

Hopefully the values and semantics are fairly obvious.

8.12.15 /vsicached/ (File caching)

The *VSICreateCachedFile()* function takes a virtual file handle and returns a new handle that caches read-operations on the input file handle. The cache is RAM based and the content of the cache is discarded when the file handle is closed. The cache is a least-recently used lists of blocks of 32KB each (default size).

This is mostly useful for files accessible through slow local/operating-system-mounted filesystems.

That is implicitly used by a number of the above mentioned file systems (namely the default one for standard file system operations, and the /vsicurl/ and other related network file systems) if the `VSI_CACHE` configuration option is set to `YES`.

The default size of caching for each file is 25 MB (25 MB for each file that is cached), and can be controlled with the `VSI_CACHE_SIZE` configuration option (value in bytes).

The `VSICachedFile` class only handles read operations at that time, and will error out on write operations.

Starting with GDAL 3.8, a `/vsicached?` virtual file system also exists to cache a particular file.

The syntax is the following one: `/vsicached?[option_i=val_i&]*file=<filename>` where each option name and value (including the value of `file`) is URL-encoded (actually, only required for the ampersand character. It might be desirable to have forward slash character uncoded). It is important that the `file` option appears at the end, so that code that tries to look for side-car files, list directory content, can work properly.

Currently supported options are:

- `chunk_size=<value>` where value is the` size of the chunk size in bytes. `KB` or `MB` suffixes can be also appended (without space after the numeric value). The maximum supported value is 1 GB.
- `cache_size=<value>` where value is the size of the cache size in bytes, for each file. `KB` or `MB` suffixes can be also appended.

Examples:

- `/vsicached?chunk_size=1MB&file=/home/even/byte.tif`
- `/vsicached?file=./byte.tif`

8.12.16 /vsicrypt/ (encrypted files)

/vsicrypt/ is a special file handler is installed that allows reading/creating/update encrypted files on the fly, with random access capabilities.

Refer to `VSIInstallCryptFileHandler()` for more details.

8.13 Feature Style Specification

Version 0.016 - 2018-12-03

8.13.1 1. Overview

This document defines the way feature style information (i.e. colors, line width, symbols, etc.) should be handled at the various levels in GDAL's vector drivers (OGR).

The following GDAL vector drivers have varying levels of support for feature styles: *DWG (libopencad)*, *DWG (Teigha)*, *DXF*, *KML (libkml)*, *MapInfo*, *MicroStation DGN v7* and *DGN v8*, *OpenJUMP JML* and *PDF*.

8.13.1.1 1.1 Style is a property of Feature object

Conceptually, the feature style should be seen as a property of a feature. Even though some systems store style information in a special attribute, in GDAL it is more consistent to see the style as a property, just the same way the geometry of a feature is also a property.

This does not prevent us from storing the style information in an attribute when writing to some formats that have no provision for styles (e.g. E00). But then at the time such a dataset is opened through GDAL, the name of the attribute that contains style information should either be specified in some metadata, or be specified by the user.

Also, in the SFCOM interface, the style information will be stored in an attribute just like the geometry is.

8.13.1.2 1.2 Feature Styles can be stored at 2 levels

The style defines the way a feature should be drawn, but it is very common to have several features that share the same style. In those cases, instead of duplicating the style information on each feature, we will provide a more efficient way to share style information.

There are two levels at which style information can be found:

- A **dataset** can have a table of pre-defined styles that can then be referred to by the layers or by the individual features. The mechanism for that is defined further down in this document.
- A **feature (OGRFeature object)** can have its own complete style definition. Alternatively, a feature can be linked to a style in the dataset's table of styles. This can save storage space when the same styles are reused often.

It should be possible to have style information stored at one or more of the various levels while working on a given dataset. The level(s) where the style is actually stored will depend on the most efficient approach for the format we are dealing with.

8.13.1.3 1.3 Drawing Tools

We define a small set of drawing tools that are used to build style definitions:

- **PEN**: For linear styles
- **BRUSH**: For filling areas
- **SYMBOL**: Point symbols
- **LABEL**: For annotations

Each drawing tool can take a number of parameters, all optional. The style syntax is built in a way that a system that cannot support all possible parameters can safely skip and ignore the parameters it does not support. This will also make it easy to extend the specification in the future without breaking existing code or applications.

A style can use a single tool, or use a combination of one or more tools. By combining the use of several tools in a style, one can build virtually any type of graphical representation. For instance, the SYMBOL tool can be used to place spaced symbols along a line. Also, the LABEL tool can be used to place text on a point, stretch it along a line, or even, by combining the PEN tool with the LABEL tool, use the line as a leader to the text label, and draw the text string on the last vertex of the line.

Of course, few systems can support all that. But the intention here is to have a style specification that is powerful and flexible enough to allow all types of formats to exchange style information with the least possible loss.

8.13.1.4 1.4 Feature attributes can be used by style definitions

In some cases, it might be useful for a style definition to refer to an attribute field on the feature for a given tool parameter's value instead of having a hardcoded value inside the style itself.

Example of this are text angle, text string, etc... these values change for every single text label, but we can share the rest of the label style at the layer level if we lookup the angle and text string in an attribute on each feature.

The syntax of the style string provides a way that any parameter value can be either a constant value, or a lookup to an attribute field.

8.13.1.5 1.5 Tool parameter units

Several parameter values can be expressed in different measurement units depending on the file format you are dealing with. For instance, some systems express line width, or text height in points, other in pixels, and others use ground units. In order to accommodate all that, all parameters can be specified in one of the following units systems:

- **g**: Map Ground Units (whatever the map coordinate units are)
- **px**: Pixels
- **pt**: Points (1/72 inch)
- **mm**: Millimeters
- **cm**: Centimeters
- **in**: Inches

Some tools will have to be provided at the GDAL client level to simplify the conversion of any value from one units system to another. This would imply that the GDAL client has to specify a map scale so that conversions from ground units to paper/pixel units can be performed.

8.13.2 2. Feature Style String

As was mentioned earlier, styles definitions will usually be stored as strings, either in a per-layer (or per-dataset) table, or directly in the features.

8.13.2.1 2.1 Examples

Here are some example style definition strings:

- A 5 pixels wide red line: `PEN(c:#FF0000,w:5px)`
- A polygon filled in blue, with a black outline: `BRUSH(fc:#0000FF);PEN(c:#000000)`
- A point symbol: `SYMBOL(c:#00FF00,id:"points.sym-45,ogr-sym-7")`

- A text label, taking the text string from the "text_attribute" attribute field: `LABEL(f:"Times New Roman", s:12pt,t:{text_attribute})"`

Here is what a style table that contains all the above styles could look like:

```
road:      PEN(c:#FF0000,w:5px)
lake:      BRUSH(fc:#0000FF);PEN(c:#000000)
campsite:  SYMBOL(c:#00FF00,id:"points.sym-45,ogr-sym-7")
label:     LABEL(f:"Times New Roman",s:12pt,t:{text_attribute})
```

Then individual features can refer to styles from the table above using the "@" character followed by the style name in their style property.

For instance, a feature with its style set to "@road" would be drawn as a red line.

8.13.2.2 2.2 Style String Syntax

Each feature object has a style property (a string):

```
<style_property> = "<style_def>" | "" | "@<style_name>" | "{<field_name>}"
```

- `<style_def>` is defined later in this section.
- An empty style string means that the feature's style is unspecified. It does not indicate that the feature is invisible – an invisible feature may be indicated using a fully transparent color, like PEN(c:#00000000).
- `@<style_name>` is a reference to a predefined style in the layer or the dataset's style table. The layer's table is looked up first, and if style_name is not found there then the dataset's table will be looked up.
- Finally, `{<field_name>}` means that the style property should be read from the specified attribute field.

The <style_def> is the real style definition. It is a combination of 1 or more style parts separated by semicolons. Each style_part uses a drawing tool to define a portion of the complete graphical representation:

```
<style_def> =     <style_part>[;<style_part>[;...]]

<style_part> =    <tool_name>([<tool_param>[,<tool_param>[,...]]])

<tool_name> =     name of a drawing tool, for now: PEN | BRUSH | SYMBOL | LABEL

<tool_param> =    <param_name>:<param_value>

<param_name> =    see list of parameters names for each drawing tool

<param_value> =   <value> | <value><units>

<value> =         "<string_value>" | <numeric_value> | {<field_name>}

<units> =         g | px | pt | mm | cm | in
```

By default, style parts are drawn in the order that they appear in the style_def string unless each part is assigned a different level parameter value (see the level parameter definition).

All drawing tool parameters are optional. So it is legal to have a style_part with an empty drawing tool parameter list (e.g. "PEN()"). For each parameter that does not have any specified value, it is up to the client application to use its own default value. This document provides advisory default values for most parameters, but it is not mandatory for an application to use those default values.

When {<field_name>} is used for a tool_param value, several options are available with respect to the units. The units can be specified after the field name as in PEN(c:#FF0000,w:{line_width}pt) or can be left unspecified as in PEN(c:#FF0000,w:{line_width}). In the first case, the default units will be points (pt), but if the attribute field line_width contains a value followed by a units abbreviation (e.g. "5px") then the units specified in the attribute fields have precedence (in this case pixels). Note that the attribute field does not have to contain a units value and probably won't in most cases; it is just an optional feature to be able to override the default units from inside an attribute field's value.

8.13.2.3 2.3 Pen Tool Parameters

Applicable geometry types:

- Point: When applied to a point, a PEN tool can only define the color and the size of the point to draw.
- Polyline: This is the most obvious case.
- Polygon: Defines the way the outline of a polygon should be drawn.

Here is the current list of PEN tool parameters. While this is sufficient to cover all the cases that we have encountered so far, new parameters might be added in the future to handle new types of graphical representation. Note again that all parameters are optional:

- `c`: **Pen Color**, expressed hexadecimal (#RRGGBB[AA])
  - [AA]: the last 2 digits define the alpha channel value, with 0 being transparent and FF being opaque. The default is FF (opaque)
  - Suggested default: black (c:#000000)
  - Example: PEN(c:#FF0000), or PEN(C:#FF0000FF)
- `w`: **Pen Width**, expressed as a numeric value with units (g, px, pt, mm, cm, in)
  - Suggested default: 1 pixel
  - Examples: PEN(c:#FF0000,w:5px), PEN(w:3pt), PEN(w:50g)
- `p`: **Pattern**. To create dash lines. A list of pen-down/pen-up distances

  Examples:
  - = PEN(c:#FF0000,w:2px,p:"4px 5px"). short-dash line
  - = PEN(c:#FF0000,w:2px,p:"10px 5px"). long-dash line
  - = PEN(c:#FF0000,w:2px,p:"10px 5px 4px 5px"). long/short dash
- `id`: **Comma-delimited list of Pen Names or Ids** For systems that identify pens with a name or an id. The names in the comma-delimited list of ids are scanned until one is recognized by the target system. Pen Ids can be either system-specific ids (see further below) or be one of the pre-defined OGR pen ids for known line patterns. The id parameter should always include one of the OGR ids at the end of the comma-delimited list of ids so that an application never has to rely on understanding system-specific ids.

  Here is the current list of OGR pen ids (this could grow in time):
  - ogr-pen-0: solid (the default when no id is provided)
  - ogr-pen-1: null pen (invisible)
  - ogr-pen-2: dash
  - ogr-pen-3: short-dash
  - ogr-pen-4: long-dash
  - ogr-pen-5: dot line

  - ogr-pen-6: dash-dot line
  - ogr-pen-7: dash-dot-dot line
  - ogr-pen-8: alternate-line (sets every other pixel)

  System-specific ids are very likely to be meaningful only to that specific system that created them. The ids should start with the system's name, followed by a dash (-), followed by whatever information is meaningful to that system (a number, a name, a filename, etc.). e.g. "mapinfo-5", or "mysoft-lines.sym-123", or "othersystems-funnyline"

  System-specific ids are allowed in order to prevent loss of information when dealing with data from systems that store line patterns in external files or that have their own pre-defined set of line styles (for instance, to do a MapInfo MIF to TAB translation without any loss.

  Examples:

  - PEN(c:#00FF00,id:"ogr-pen-0") - simple solid line
  - PEN(c:#00FF00,id:"mapinfo-5,ogr-pen-7") - corresponds to MapInfo's Pen #5, and a system that can't understand MapInfo pens falls back on the default "ogr-pen-7" pen (dot-dot line).

- `cap`: **Pen Cap** - Set the shape of end points of lines.
  - "cap:b" - Butt: The ends of the line don't extend beyond the end points. This is the default.
  - "cap:r" - Round: Terminate lines with a circle whose diameter is equal to the line width.
  - "cap:p" - Projecting: Similar to Butt, but the ends of the line extend by half of line width beyond the end points.
- `j`: **Pen Join** - Set the shape of the join point (vertex) of lines.
  - "j:m" - Miter: Extend the outer edge of the lines until they touch. This is the default.
  - "j:r" - Rounded: Join lines with an arc whose center is at the join point and whose diameter is equal to the line width.
  - "j:b" - Bevel: Join the lines with butt end caps and fill the resulting triangular notch at the join position.
- `dp`: **Perpendicular Offset**, expressed as a numeric value units (g, px, pt, mm, cm, in)

  Offset from the line center. If the offset is negative then the pen will be drawn left of the main segment and right otherwise.
- `l`: **Priority Level** - Numeric value defining the order in which style parts should be drawn.

  Lower priority style parts are drawn first, and higher priority ones are drawn on top. If priority level is unspecified, the default is 1.

8.13.2.4 2.4 Brush Tool Parameters

Applicable geometry types:

- Point: Not applicable.
- Polyline: Not applicable.
- Polygon: Defines the way the surface of a polygon is filled.

Here is the current list of BRUSH tool parameters. Note again that that this list may be extended in the future, and all parameters are optional:

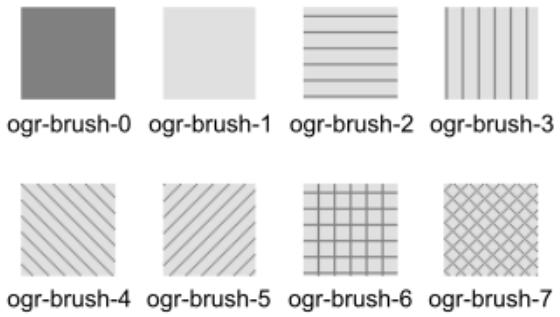
- `fc`: **Brush ForeColor**, expressed in hexadecimal (#RRGGBB[AA]). Used for painting the brush pattern itself.
  - [AA]: the last 2 digits define the alpha channel value, with 0 being transparent and FF being opaque. The default is FF (opaque)

  - Suggested default: 50% grey (c:#808080)
  - Example: BRUSH(fc:#FF0000)

- `bc`: **Brush BackColor**, expressed in hexadecimal (#RRGGBB[AA]). Used for painting the area behind the brush pattern.
  - [AA]: the last 2 digits define the alpha channel value, with 0 being transparent and FF being opaque. The default is FF (opaque)
  - Suggested default: transparent (c:#FFFFFF00)
  - Example: BRUSH(fc:#FF0000,bc:#FFEEDD)
- `id`: **Brush Name or Brush Id** - Comma-delimited list of brush names or ids. The names in the comma-delimited list of ids are scanned until one is recognized by the target system.

  Brush Ids can be either system-specific ids (see further below) or be one of the pre-defined OGR brush ids for well known brush patterns. The id parameter should always include one of the OGR ids at the end of the comma-delimited list of ids so that an application never has to rely on understanding system-specific ids.

  Here is the current list of OGR brush ids (this could grow over time):




```
BRUSH(fc:#808080,bc:#e0e0e0,id:"ogr-brush-n")
```

  - ogr-brush-0: solid foreground color (the default when no id is provided)
  - ogr-brush-1: null brush (transparent - no fill, irrespective of fc or bc values
  - ogr-brush-2: horizontal hatch
  - ogr-brush-3: vertical hatch
  - ogr-brush-4: top-left to bottom-right diagonal hatch
  - ogr-brush-5: bottom-left to top-right diagonal hatch
  - ogr-brush-6: cross hatch
  - ogr-brush-7: diagonal cross hatch

  Like with Pen Ids, system-specific brush ids are very likely to be meaningful only to that specific system that created them. The ids should start with the system's name, followed by a dash (-), followed by whatever information is meaningful to that system (a number, a name, a filename, etc.).

  The following conventions will be used for common system-specific brush ids:

  - "bmp-filename.bmp" for Windows BMP patterns

  Other conventions may be added in the future (such as vector symbols, WMF, etc).

- `a`: **Angle** - Rotation angle (in degrees, counterclockwise) to apply to the brush pattern.

- `s`: **Size or Scaling Factor** - Numeric value with or without units.

  If units are specified, then this value is the absolute size to draw the brush or symbol. If no units are specified then it is taken as a scaling factor relative to the symbol's default size.

- `dx`, `dy`: **Spacing**, expressed as a numeric value with units (g, px, pt, mm, cm, in)

  If filling an area using point symbols, these values will define the spacing to use between them. "dx" is the horizontal distance between the center of two adjacent symbols and "dy" is the vertical distance. The default is to use the symbol's bounding box width and height for dx and dy respectively.

- `l`: **Priority Level** - Numeric value defining the order in which style parts should be drawn.

  Lower priority style parts are drawn first, and higher priority ones are drawn on top. If priority level is unspecified, the default is 1.

8.13.2.5 2.5 Symbol Tool Parameters

Applicable geometry types:

- Point: Place a symbol at the point's location
- Polyline: Place symbols along the polyline, either at each vertex, or equally spaced.
- Polygon: Place the symbols on the outline of the polygon.

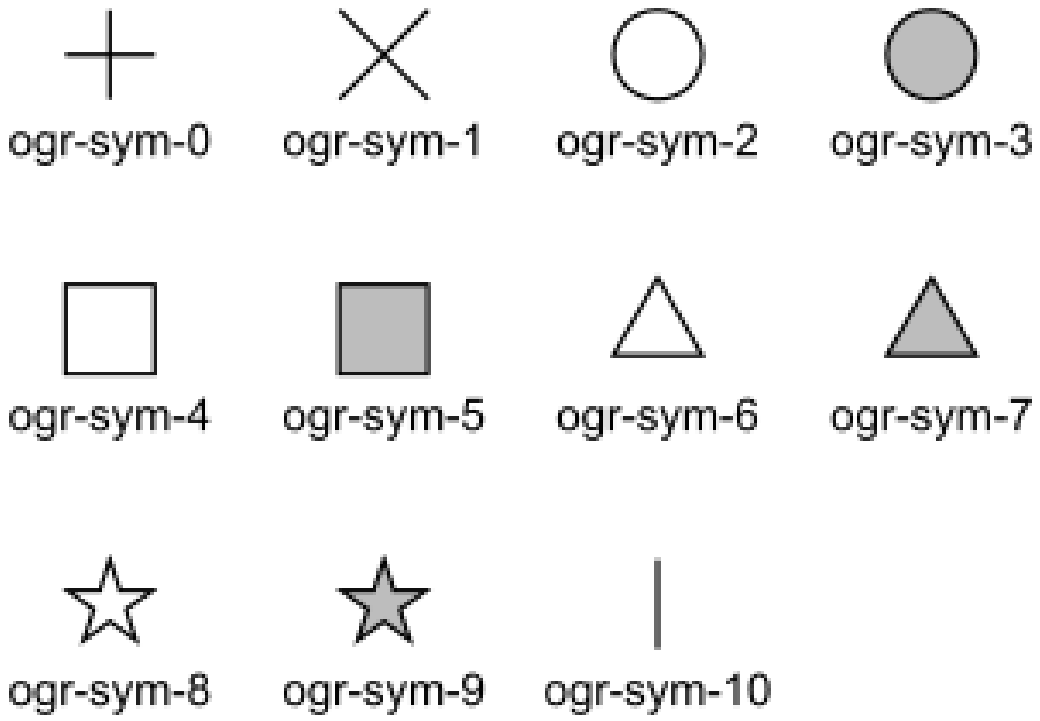
Here is the current list of SYMBOL tool parameters. Note again that that this list may be extended in the future, and all parameters are optional:

- `id`: **Symbol Name or Id** - Comma-delimited list of symbol names or ids.

  The names in the comma-delimited list of ids are scanned until one is recognized by the target system.

  Symbol Ids can be either system-specific ids (see further below) or be one of the pre-defined OGR symbol ids for well known symbols. The id parameter should always include one of the OGR ids at the end of the comma-delimited list of ids so that an application never has to rely on understanding system-specific ids.

  Here is the current list of OGR symbol ids (this could grow over time):



- ogr-sym-0: cross (+)
- ogr-sym-1: diagcross (X)
- ogr-sym-2: circle (not filled)
- ogr-sym-3: circle (filled)
- ogr-sym-4: square (not filled)
- ogr-sym-5: square (filled)
- ogr-sym-6: triangle (not filled)
- ogr-sym-7: triangle (filled)
- ogr-sym-8: star (not filled)
- ogr-sym-9: star (filled)
- ogr-sym-10: vertical bar (can be rotated using angle attribute to produce diagonal bar)

Like with Pen Ids, system-specific symbol ids are very likely to be meaningful only to that specific system that created them. The ids should start with the system's name, followed by a dash (-), followed by whatever information is meaningful to that system (a number, a name, a filename, etc.).

The following conventions will be used for common system-specific symbol ids:

- "bmp-filename.bmp" for Windows BMP symbols
- "font-sym-%d" for a font symbols, where %d is a glyph number inside a font, font family is defined by **f** style field.

Other conventions may be added in the future (such as vector symbols, WMF, etc.)

- a: **Angle** - Rotation angle (in degrees, counterclockwise) to apply to the symbol.
- c: **Symbol Color**, expressed in hexadecimal (#RRGGBB[AA])
  - [AA]: the last 2 digits define the alpha channel value, with 0 being transparent and FF being opaque. The default is FF (opaque)
  - Suggested default symbol color: black (c:#000000)
  - Example

```
SYMBOL(c:#FF0000)
```

- o: **Symbol Outline Color**, expressed in hexadecimal (#RRGGBB[AA]).

  If this parameter is set, an additional halo or border of this color is drawn around the symbol.
- s: **Size or Scaling Factor** - Numeric value with or without units.

  If units are specified, then this value is the absolute size to draw the symbol. If no units are specified then it is taken as a scaling factor relative to the symbol's default size.
- dx, dy: **X and Y offset**, of the symbol's insertion point, expressed as a numeric value with units (g, px, pt, mm, cm, in)

  Applies to point geometries, and to symbols placed at each vertex of a polyline.
- ds, dp, di: **Spacing parameters** for symbols spaced along a line, expressed as a numeric value with units (g, px, pt, mm, cm, in).
  - ds is the step to use when placing symbols along the line. By default, symbols applied to a feature with a line geometry are placed at each vertex, but setting "ds" triggers the placement of symbols at an equal distance along the line. "ds" has no effect for a feature with a point geometry.
  - dp can be used together with "ds" to specify the perpendicular distance between the symbols' center and the line along which they're placed.
  - di can be used to specify an initial offset from the beginning of the line.
  - Example

```
SYMBOL(id:123, s:5, di:5px, ds:50px)
```

- l: **Priority Level** - Numeric value defining the order in which style parts should be drawn.

  Lower priority style parts are drawn first, and higher priority ones are drawn on top. If priority level is unspecified, the default is 1.
- f: **Font Name** - Comma-delimited list of fonts names.

  Works like the CSS font-family property: the list of font names is scanned until a known font name is encountered.

  Example

```
SYMBOL(c:#00FF00,s:12pt,id:"font-sym-75,ogr-sym-9",f:"MapInfo_Cartographic")
```

8.13.2.6 2.6 Label Tool Parameters

Applicable geometry types:

- Point: Place a text label at the point's location
- Polyline: Place text along the polyline.

- Polygon: Place a label at the centroid of the polygon. All parameters behave exactly as if the geometry was a point located at the polygon's centroid.

Here is the current list of LABEL tool parameters. Note again that that this list may be extended in the future, and all parameters are optional:

- `f`: **Font Name** - Comma-delimited list of fonts names.

  Works like the CSS font-family property: the list of font names is scanned until a known font name is encountered.

  Example

```
LABEL(f:"Noto Sans, Helvetica", s:12pt, t:"Hello World!")
```

- `s`: **Font Size**, expressed as a numeric value with units (g, px, pt, mm, cm, in).

  In the CAD world, font size, or "text height", determines the height of a capital letter - what typographers call "cap height". But in the worlds of typesetting, graphics and cartography, font size refers to the "em height" of the font, which is taller than the cap height. This means that text assigned a height of 1 inch in a DXF file will look larger (often about 45% larger) than 72-point text in a PDF file or MapInfo map. At present, GDAL vector drivers treat the "s:" style string value as whichever font size measurement (cap height or em height) is used natively by that format, which may result in incorrect text sizing when using the ogr2ogr tool. This parameter could be subject to clearer specification in the future.

- `t`: **Text String** - Can be a constant string, or a reference to an attribute field's value.

  If a double-quote character or backslash () character is present in the string, it is escaped with a backslash character before it.

  Examples

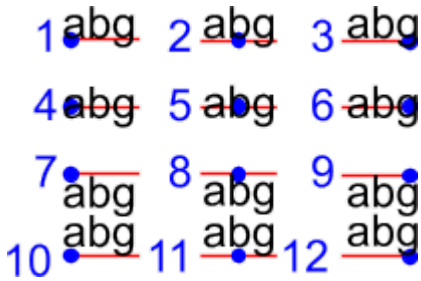
```
LABEL(f:"Arial, Helvetica", s:12pt, t:"Hello World!")
LABEL(f:"Arial, Helvetica", s:12pt, t:"Hello World with escaped \\"quotes\" and \\\
↪backslash!")
LABEL(f:"Arial, Helvetica", s:12pt, t:{text_attribute})
```

- `a`: **Angle** - Rotation angle (in degrees, counterclockwise).
- `c`: **Text Foreground Color**, expressed in hexadecimal (#RRGGBB[AA]) Suggested default: black (c:#000000)
- `b`: **Text Background Color** - Color of the filled box to draw behind the label, expressed in hexadecimal (#RRGGBB[AA]). No box drawn if not set.
- `o`: **Text Outline Color** - Color of the text outline (halo in MapInfo terminology), expressed in hexadecimal (#RRGGBB[AA]). No outline if not set.
- `h`: **Shadow Color** - Color of the text shadow, expressed in hexadecimal (#RRGGBB[AA]). No shadow if not set.
- `w`: **Stretch** - The stretch factor changes the width of all characters in the font by the given percentage. For example, a setting of 150 results in all characters in the font being stretched to 150% of their usual width. The default stretch factor is 100.
- `m`: **Label Placement Mode** - How the text is drawn relative to the feature's geometry.
  - "m:p" - The default. A simple label is attached to a point or to the first vertex of a polyline.
  - "m:l" - Text is attached to the last vertex of a polyline. A PEN tool can be combined with this LABEL tool to draw the polyline as a leader to the label.
  - "m:s" - Stretch the text string along a polyline, with an equal spacing between each character.
  - "m:m" - Place text as a single label at the middle of a polyline (based on total line length).
  - "m:w" - One word per line segment in a polyline.

  - "m:h" - Every word of text attached to polyline is placed horizontally in its segment, anchor point is a center of segment.
  - "m:a" - Every word of text attached to polyline is stretched to fit the segment of polyline and placed along that segment. The anchor point is a start of a segment.

- `p`: **Anchor Position** - A value from 1 to 12 defining the label's position relative to the point to which it is attached

  There are four vertical alignment modes: *baseline*, *center*, *top* and *bottom*; and three horizontal modes: *left*, *center* and *right*.

  Currently, the precise interpretation of these values (for example, whether accents on uppercase letters sit above or below the alignment point with p:7) differs from file format to file format. This parameter could be subject to clearer specification in the future.

- `dx`, `dy`: **X and Y offset** of the label's insertion point, expressed as a numeric value with units (g, px, pt, mm, cm, in).

  Applies to text placed on a point, or at each vertex of a polyline.

- `dp`: **Perpendicular Offset** for labels placed along a line, expressed as a numeric value with units (g, px, pt, mm, cm, in).

  "dp" specifies the perpendicular distance between the label and the line along which it is placed. If the offset is negative then the label will be shifted left of the main segment, and right otherwise.

- `bo`: **Bold** - Set to 1 for bold text. Set to 0 or omitted otherwise.
- `it`: **Italic** - Set to 1 for italic text. Set to 0 or omitted otherwise.
- `un`: **Underline** - Set to 1 for underlined text. Set to 0 or omitted otherwise.
- `st`: **Strikethrough** - Set to 1 for struck-through text. Set to 0 or omitted otherwise.
- `l`: **Priority Level** - Numeric value defining the order in which style parts should be drawn.

  Lower priority style parts are drawn first, and higher priority ones are drawn on top. If priority level is unspecified, the default is 1.

8.13.2.7 2.7 Styles Table Format

For file formats that support tables of styles, then the predefined styles would be stored in that format.

For file formats that do not support tables of styles, then the style table could be stored in a text file with a .ofs (OGR Feature Styles) extension and the same basename as the dataset. This would apply to formats like Esri Shapefile.

Here is an example of a .ofs file:

```
#OFS-Version: 1.0
#StyleField: "style"

DefaultStyle: PEN(c:#000000)
```

(continues on next page)

(continued from previous page)

```
road:      PEN(c:#FF0000,w:5px)
lake:      BRUSH(fc:#0000FF);PEN(c:#000000)
campsite:  SYMBOL(c:#00FF00,id:"points.sym-45,ogr-sym-7")
label:     LABEL(f:"Times New Roman",s:12pt,t:{text_attribute})
```

The first line is a signature with a version number, which must be present.

The second line (StyleField: "style") is the name of the attribute field in which the Feature Style String is stored for each object in the corresponding layer. This is optional, if not set, then the objects in the layer will all share the same style defined in DefaultStyle.

The third line (DefaultStyle:...) defines the style that applies by default to all objects that have no explicit style.

Then the list of style definitions follow.

8.13.2.8 2.8 Using OGR SQL to transfer the style between the data sources

You can use the **OGR_STYLE** special field to extract the feature level style, and ogr2ogr can be used to transfer the style string between data sources according to the following example:

```
ogr2ogr -f "ESRI Shapefile" -sql "select *, OGR_STYLE from rivers" rivers.shp rivers.tab
```

Without specifying the length of the style field, the output driver may truncate the length to a default value. Therefore it may be necessary to specify the target length manually, like:

```
ogr2ogr -f "ESRI Shapefile" -sql "select *, CAST(OGR_STYLE AS character(255)) from rivers
↪" rivers.shp rivers.tab
```

OGR is aware of using the OGR_STYLE field if it exists, and OGRFeature::GetStyleString will return the value of this field if no style string has been specified programmatically.

8.13.3 3. OGR Support Classes

The *OGRFeature* class has member functions *OGRFeature::GetStyleString()*, *OGRFeature::SetStyleString()* and *OGRFeature::SetStyleStringDirectly()* which may be used to interact with a feature's style string as a C-style string. Additionally, there are *OGRFeature::GetStyleTable()*, *OGRFeature::SetStyleTable()* and *OGRFeature::SetStyleTableDirectly()* for managing style tables as instances of the *OGRStyleTable* class.

The *OGRLayer* and *GDALDataset* classes also have *OGRLayer::GetStyleTable()*, *OGRLayer::SetStyleTable()* and *OGRLayer::SetStyleTableDirectly()* member functions.

To parse style strings, the *OGRStyleMgr* class is used. Each style tool in the string is accessed as an instance of the *OGRStyleTool* class. Lastly, four helper classes exist, one for each tool (`OGRStylePen`, `OGRStyleBrush`, `OGRStyleSymbol`, `OGRStyleLabel`), with each available parameter represented by a getter and setter member function. To understand these classes better, it may be useful to read the ogr_featurestyle.h and ogrfeaturestyle.cpp code files.

Here is some example C++ code:

```
OGRStyleTable oStyleTable;

OGRStyleMgr *poStyleMgr = new OGRStyleMgr(&oStyleTable);
```

(continues on next page)

(continued from previous page)

```
// Create a new style in the style table by specifying the whole style string

if (poStyleMgr->AddStyle("@Name","PEN(c:#123456;w:10px);BRUSH(c:#345678)"))
{
  poStyleMgr->SetFeatureStyleString(poFeature,"@Name",TRUE)
  // or
  poStyleMgr->SetFeatureStyleString(poFeature,"PEN(c:#123456,w:10px);BRUSH(c:#345678)",
↪FALSE)
}

oStyleTable->SaveStyleTable("ttt.tbl");


// Create a new style in the style table by specifying each tool (part) as a string

poStyleMgr->InitStyleString();
poStyleMgr->AddPart("PEN(c:#123456,w:10px)");
poStyleMgr->AddPart("BRUSH(c:345678)");
poStyleMgr->AddStyle("@Name");
poStyleMgr->SetFeatureStyleString(poFeature,"@Name",TRUE);

oStyleTable->SaveStyleTable("ttt.tbl");


// Create a new style in the style table using the style tool helper classes

OGRStyleTool *poStylePen = new OGRStylePen;

poStylePen->SetColor("#123456");
poStylePen->SetUnit(OGRSTUPixel);
poStylePen->SetWidth(10.0);
poStyleMgr->AddPart(poStylePen);

delete poStylePen;


// Reading a style

OGRStyleTool *poStyleTool;

poStyleMgr->GetStyleString(poFeature);

for (int iPart = 0; iPart < poStyleMgr->GetPartCount(); iPart++)
{
   poStyleTool = GetPart(iPart);
   switch (poStyleTool->GetType())
   {
      case OGRSTCPen:
         poStylePen = (OGRStylePen *)poStyleTool;
         pszColor = poStylePen->Color(bDefault);
         if (bDefault == FALSE)
           poStylePen->GetRGBFromString(pszColor, nRed, nGreen,
```

(continues on next page)

(continued from previous page)

```
                                    nBlue, nTrans);
        else
          // Color not defined

        dfWidth = poStylePen->Width(bDefault);
        if (bDefault == FALSE)
          // Use dfWidth
        else
          // dfWidth not defined

       :
       :
   }
}
```

8.14 Coordinate epoch support

Added in version 3.4.

8.14.1 Dynamic CRS and coordinate epoch

This document is intended to document the support for coordinate epoch, linked to dynamic CRS.

In a dynamic CRS, coordinates of a point on the surface of the Earth may change with time. To be unambiguous the coordinates must always be qualified with the epoch at which they are valid. The coordinate epoch is not necessarily the epoch at which the observation was collected.

Examples of dynamic CRS are `WGS 84 (G1762)`, `ITRF2014`, `ATRF2014`.

The generic EPSG:4326 WGS 84 CRS is also considered dynamic, although it is not recommended to use it due to being based on a datum ensemble whose positional accuracy is 2 meters, but prefer one of its realizations, such as WGS 84 (G1762)

The *`OGRSpatialReference::IsDynamic()`* method can be used to test if a CRS is a dynamic one.

The *`OGRSpatialReference::SetCoordinateEpoch()`* and *`OGRSpatialReference::GetCoordinateEpoch()`* methods can be used to set/retrieve a coordinate epoch associated with a CRS. The coordinate epoch is expressed as a decimal year (e.g. 2021.3 for April 21, 2021).

Formally, the coordinate epoch of an observation belongs to the observation. However, almost all formats do not allow for storing per-observation epoch, and typical usage is a set of observations with the same epoch. Therefore we store the epoch as property of the CRS, and assume that it is valid for every observation. This choice eases processing, storage and format complexity for most usage. For now, this means that a dataset containing observations or points with different epochs cannot be handled.

For vector formats, per-geometry coordinate epoch could also make sense, but as most formats only support a per-layer CRS, we also for now limit support of coordinate epoch at the layer level. The underlying coordinate transformation mechanics can support per-vertex coordinate epoch.

8.14.2 Support in raster and vector formats

At time of writing, no formats handled by GDAL/OGR have a standardized way of encoding a coordinate epoch. We consequently have made choices how to encode it, with the aim of being as much as possible backward compatible with existing readers. Those encodings might change if corresponding official specifications evolve to take this concept into account. The coordinate epoch is only written when attached to the SRS of the layer/dataset that is created.

8.14.2.1 FlatGeoBuf

The coordinate epoch is encoded as a WKT:2019 string using the `EPOCH` subnode of the COORDINATEMETADATA construct, set in the `Crs.wkt` header field of the FlatGeoBuf file.

```
COORDINATEMETADATA[
    GEOGCRS["WGS 84 (G1762)",
        DYNAMIC[FRAMEEPOCH[2005.0]],
        DATUM["World Geodetic System 1984 (G1762)",
          ELLIPSOID["WGS 84",6378137,298.257223563,LENGTHUNIT["metre",1.0]]
        ],
        CS[ellipsoidal,3],
          AXIS["(lat)",north,ANGLEUNIT["degree",0.0174532925199433]],
          AXIS["(lon)",east,ANGLEUNIT["degree",0.0174532925199433]],
          AXIS["ellipsoidal height (h)",up,LENGTHUNIT["metre",1.0]]
    ],
    EPOCH[2016.47]
]
```

> **Note**
>
> Such construct will not be understood by GDAL < 3.4, but if the CRS has an associated EPSG code, this will not cause issues in those older GDAL versions.

8.14.2.2 GeoPackage vector/raster

Each vector/raster table which has an associated coordinate epoch encodes it in the `epoch` column of the `gpkg_spatial_ref_sys` table, using an extended version of the CRS WKT extension (https://github.com/opengeospatial/geopackage/pull/600).

8.14.2.3 GeoTIFF

The coordinate epoch is encoded as a new GeoTIFF GeoKey, `CoordinateEpochGeoKey` of code 5120 and type DOUBLE.

```
Geotiff_Information:
   Version: 1
   Key_Revision: 1.0
   Tagged_Information:
      ModelTiepointTag (2,3):
         0                0                0
         440720           3751320          0
      ModelPixelScaleTag (1,3):
         60               60               0
      End_Of_Tags.
   Keyed_Information:
      GTModelTypeGeoKey (Short,1): ModelTypeProjected
      GTRasterTypeGeoKey (Short,1): RasterPixelIsArea
      GTCitationGeoKey (Ascii,22): "WGS 84 / UTM zone 11N"
      GeogCitationGeoKey (Ascii,7): "WGS 84"
      GeogAngularUnitsGeoKey (Short,1): Angular_Degree
      ProjectedCSTypeGeoKey (Short,1): PCS_WGS84_UTM_zone_11N
      ProjLinearUnitsGeoKey (Short,1): Linear_Meter
```

(continues on next page)

(continued from previous page)

```
    CoordinateEpochGeoKey (Double,1): 2021.3
    End_Of_Keys.
  End_Of_Geotiff.
```

8.14.2.4 JPEG2000

GeoJP2 boxes use the above mentioned GeoTIFF encoding.

8.14.2.5 Persistent Auxiliary Metadata (.aux.xml)

The coordinate epoch is encoded as `coordinateEpoch` attribute of the `SRS` element.

```
<PAMDataset>
  <SRS dataAxisToSRSAxisMapping="1,2" coordinateEpoch="2021.3">PROJCS["WGS 84 / UTM zone␣
↪11N",GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563,
↪AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",0,AUTHORITY[
↪"EPSG","8901"]],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],AUTHORITY[
↪"EPSG","4326"]],PROJECTION["Transverse_Mercator"],PARAMETER["latitude_of_origin",0],
↪PARAMETER["central_meridian",-117],PARAMETER["scale_factor",0.9996],PARAMETER["false_
↪easting",500000],PARAMETER["false_northing",0],UNIT["metre",1,AUTHORITY["EPSG","9001
↪"]],AXIS["Easting",EAST],AXIS["Northing",NORTH],AUTHORITY["EPSG","32611"]]</SRS>
  <!-- snip -->
</PAMDataset>
```

8.14.2.6 GDAL VRT

The coordinate epoch is encoded as `coordinateEpoch` attribute of the `SRS` element.

```
<VRTDataset rasterXSize="20" rasterYSize="20">
  <SRS dataAxisToSRSAxisMapping="1,2" coordinateEpoch="2021.3">PROJCS["WGS 84 / UTM zone␣
↪11N",GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563,
↪AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",0,AUTHORITY[
↪"EPSG","8901"]],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],AUTHORITY[
↪"EPSG","4326"]],PROJECTION["Transverse_Mercator"],PARAMETER["latitude_of_origin",0],
↪PARAMETER["central_meridian",-117],PARAMETER["scale_factor",0.9996],PARAMETER["false_
↪easting",500000],PARAMETER["false_northing",0],UNIT["metre",1,AUTHORITY["EPSG","9001
↪"]],AXIS["Easting",EAST],AXIS["Northing",NORTH],AUTHORITY["EPSG","32611"]]</SRS>
  <!-- snip -->
</VRTDataset>
```

8.14.3 Support in utilities

gdalinfo and **ogrinfo** report the coordinate epoch, when attached to a dataset/layer SRS.

gdal_edit.py has a `-a_coord_epoch` option to define the epoch of a dataset in place.

gdal_translate and **ogr2ogr** have a `-a_coord_epoch` option to be used together with `-a_srs`, and otherwise preserve the coordinate epoch in the output SRS from the source SRS when no SRS related options are specified.

gdalwarp and **ogr2ogr** have a `-s_coord_epoch` option to be used together with `-s_srs` (resp. `-t_coord_epoch` option to be used together with `-t_srs`) to override/set the coordinate epoch of the source (resp. target) CRS.

Before PROJ 9.4, `-s_coord_epoch` and `-t_coord_epoch` were mutually exclusive, due to lack of support for transformations between two dynamic CRS.

gdalwarp preserves the coordinate epoch in the output SRS when appropriate.

8.14.4 Support in coordinate transformation

The *OGRCoordinateTransformation* class can perform time-dependent transformations between a static and dynamic CRS based on the coordinate epoch passed per vertex.

It can also take into account the coordinate epoch associated with a dynamic CRS, when doing time-dependent transformations between a static and dynamic CRS. The *OGR_CT_USE_SRS_COORDINATE_EPOCH* configuration option can be set to NO to disable using the coordinate epoch associated with the source or target CRS.

If a per-vertex time is specified, it overrides the one associated with the CRS.

8.15 Configuration options

This page discusses runtime configuration options for GDAL. These are distinct from options to the build-time configure script. Runtime configuration options apply on all platforms, and are evaluated at runtime. They can be set programmatically, by commandline switches or in the environment by the user.

Configuration options are normally used to alter the default behavior of GDAL/OGR drivers and in some cases the GDAL/OGR core. They are essentially global variables the user can set.

8.15.1 How to set configuration options?

One example of a configuration option is the *GDAL_CACHEMAX* option. It controls the size of the GDAL block cache, in megabytes. It can be set in the environment on Unix (bash/bourne) shell like this:

```
export GDAL_CACHEMAX=64
```

In a DOS/Windows command shell it is done like this:

```
set GDAL_CACHEMAX=64
```

It can also be set on the commandline for most GDAL and OGR utilities with the `--config` switch, though in a few cases these switches are not evaluated in time to affect behavior.

```
gdal_translate --config GDAL_CACHEMAX 64 in.tif out.tif
```

Since GDAL 3.9, it is also possible to set a config option in a more conventional way by using a single <NAME>``=``<VALUE> command line string instead of having <NAME> and <VALUE> as two space-separated strings.

```
gdal_translate --config GDAL_CACHEMAX=64 in.tif out.tif
```

In C/C++ configuration switches can be set programmatically with *CPLSetConfigOption()*:

```
#include "cpl_conv.h"
...
    CPLSetConfigOption( "GDAL_CACHEMAX", "64" );
```

Normally a configuration option applies to all threads active in a program, but they can be limited to only the current thread with *CPLSetThreadLocalConfigOption()*

```
CPLSetThreadLocalConfigOption( "GTIFF_DIRECT_IO", "YES" );
```

For boolean options, the values YES, TRUE or ON can be used to turn the option on; NO, FALSE or OFF to turn it off.

8.15.2 How to detect if the passed configuration option is known to GDAL

By default GDAL will not warn if the name of the configuration option is unknown. Starting with GDAL 3.11, if you set the *CPL_DEBUG* configuration option to `ON` (or any value that is not `OFF`, `FALSE`, `NO`), a GDAL warning will be emitted for unknown configuration options.

```
$ gdalinfo --config BAD_OPTION=TEST --debug on --version
Warning 1: CPLSetConfigOption() called with key=BAD_OPTION, which is unknown to GDAL
[...]
```

8.15.3 GDAL configuration file

Added in version 3.3.

On driver registration, loading of configuration is attempted from a set of predefined files.

The following locations are tried by *CPLLoadConfigOptionsFromPredefinedFiles()*:

- the location pointed by the environment variable (or configuration option) *GDAL_CONFIG_FILE* is attempted first. If it is set, the next steps are not attempted
- for Unix builds, the location pointed by ${sysconfdir}/gdal/gdalrc is first attempted (where ${sysconfdir} evaluates to ${prefix}/etc, unless the `--sysconfdir` switch of ./configure has been invoked). Then $(HOME)/.gdal/gdalrc is tried, potentially overriding what was loaded with the sysconfdir
- for Windows builds, the location pointed by $(USERPROFILE)/.gdal/gdalrc is attempted.

A configuration file is a text file in a .ini style format. Lines starting with # are comment lines.

The file may contain a `[configoptions]` section, that lists configuration options and their values.

Example:

```
[configoptions]
# set BAR as the value of configuration option FOO
FOO=BAR
```

Configuration options set in the configuration file can later be overridden by calls to *CPLSetConfigOption()* or *CPLSetThreadLocalConfigOption()*, or through the `--config` command line switch.

The value of environment variables set before GDAL starts will be used instead of the value set in the configuration files, unless, starting with GDAL 3.6, the configuration file starts with a `[directives]` section that contains a `ignore-env-vars=yes` entry.

```
[directives]
# ignore environment variables. Take only into account the content of the
# [configoptions] section, or ones defined programmatically with
# CPLSetConfigOption / CPLSetThreadLocalConfigOption.
ignore-env-vars=yes
```

Starting with GDAL 3.5, a configuration file can also contain credentials (or more generally options related to a virtual file system) for a given path prefix, that can also be set with *VSISetPathSpecificOption()*. Credentials should be put under a `[credentials]` section, and for each path prefix, under a relative subsection whose name starts with "[." (e.g. "[.some_arbitrary_name]"), and whose first key is "path".

Example:

```
[credentials]
```

(continues on next page)

(continued from previous page)

```
[.private_bucket]
path=/vsis3/my_private_bucket
AWS_SECRET_ACCESS_KEY=...
AWS_ACCESS_KEY_ID=...

[.sentinel_s2_l1c]
path=/vsis3/sentinel-s2-l1c
AWS_REQUEST_PAYER=requester
```

8.15.4 Global configuration options

8.15.4.1 Logging

- **CPL_CURL_VERBOSE=[YES/NO]:** Set to "YES" to get the curl library to display a lot of verbose information about its operations. Very useful for libcurl and/or protocol debugging and understanding.
- **CPL_DEBUG=[ON/OFF/<PREFIX>]:** This may be set to ON, OFF or specific prefixes. If it is ON, all debug messages are reported to stdout. If it is OFF or unset no debug messages are reported. If it is set to a particular value, then only debug messages with that "type" value will be reported. For instance debug messages from the HFA driver are normally reported with type "HFA" (seen in the message). At the commandline this can also be set with --debug <value> as well as with --config CPL_DEBUG <value>.
- **CPL_LOG=<path>:** This is used for setting the log file path.
- **CPL_LOG_ERRORS=[ON/OFF]:** Set to "ON" for printing error messages. Use together with "CPL_LOG" for directing them into a file.
- **CPL_TIMESTAMP=[ON/OFF]:** Set to "ON" to add timestamps to CPL debug messages (so assumes that *CPL_DEBUG* is enabled)
- **CPL_MAX_ERROR_REPORTS=value:**
- **CPL_ACCUM_ERROR_MSG=value:**
- **GDAL_NAME_AND_SHAME=[ON/OFF]:** See cloud_phone_home for details.

8.15.4.2 Performance and caching

- **GDAL_NUM_THREADS=[ALL_CPUS/<integer>]:** Sets the number of worker threads to be used by GDAL operations that support multithreading. The default value depends on the context in which it is used.
- **GDAL_CACHEMAX=<size>:** Defaults to `5%`. Controls the default GDAL raster block cache size. When blocks are read from disk, or written to disk, they are cached in a global block cache by the *GDALRasterBlock* class. Once this cache exceeds *GDAL_CACHEMAX* old blocks are flushed from the cache. This cache is mostly beneficial when needing to read or write blocks several times. This could occur, for instance, in a scanline oriented input file which is processed in multiple rectangular chunks by **gdalwarp**. If its value is small (less than 100000), it is assumed to be measured in megabytes, otherwise in bytes. Alternatively, the value can be set to "X%" to mean X% of the usable physical RAM. Since GDAL 3.11, the value of *GDAL_CACHEMAX* may specify the units directly (e.g., "500MB", "2GB"). Note that this value is only consulted the first time the cache size is requested. To change this value programmatically during operation of the program it is better to use *GDALSetCacheMax()* (always in bytes) or or *GDALSetCacheMax64()*. The maximum practical value on 32 bit OS is between 2 and 4 GB. It is the responsibility of the user to set a consistent value.
- **GDAL_FORCE_CACHING=[YES/NO]:** Defaults to `NO`. When set to YES, *GDALDataset::RasterIO()* and *GDALRasterBand::RasterIO()* will use cached IO (access block by block through *GDALRasterBand::IReadBlock()* API) instead of a potential driver-specific implementation of IRasterIO(). This will only have an effect on drivers that specialize IRasterIO() at the dataset or raster band level, for example JP2KAK, NITF, HFA, WCS, ECW, MrSID, and JPEG.

- **GDAL_BAND_BLOCK_CACHE=[AUTO/ARRAY/HASHSET]:** Defaults to `AUTO`. Controls whether the block cache should be backed by an array or a hashset. By default (`AUTO`) the implementation will be selected based on the number of blocks in the dataset. See rfc-26 for more information.
- **GDAL_MAX_DATASET_POOL_SIZE=value:** Defaults to `100`. Used by gdalproxypool.cpp Number of datasets that can be opened simultaneously by the GDALProxyPool mechanism (used by VRT for example). Can be increased to get better random I/O performance with VRT mosaics made of numerous underlying raster files. Be careful: on Linux systems, the number of file handles that can be opened by a process is generally limited to 1024. This is currently clamped between 2 and 1000.
- **GDAL_MAX_DATASET_POOL_RAM_USAGE=value:** (GDAL >= 3.7) Limit the RAM usage of opened datasets in the GDALProxyPool. The value can also be suffixed with `MB` or `GB` to respectively express it in megabytes or gigabytes. The default value is 25% of the usable physical RAM minus the *GDAL_CACHEMAX* value.
- **GDAL_SWATH_SIZE=value:** Defaults to `1/4` of the maximum block cache size (``GDAL_CACHEMAX``). Used by rasterio.cpp Size of the swath when copying raster data from one dataset to another one (in bytes). Should not be smaller than *GDAL_CACHEMAX*.
- **GDAL_DISABLE_READDIR_ON_OPEN=[TRUE/FALSE/EMPTY_DIR]:** Defaults to `FALSE`. By default (FALSE), GDAL establishes a list of all the files in the directory of the file passed to *GDALOpen()*. This can result in speed-ups in some use cases, but also to major slow-downs when the directory contains thousands of other files. When set to TRUE, GDAL will not try to establish the list of files. The number of files read can also be limited by *GDAL_READDIR_LIMIT_ON_OPEN*. If set to EMPTY_DIR, only the file that is being opened will be seen when a GDAL driver will request sibling files, so this is a way to disable loading side-car/auxiliary files.
- **GDAL_READDIR_LIMIT_ON_OPEN=value:** Defaults to `1000`. Sets the maximum number of files to scan when searching for sidecar files in *GDALOpen()*.
- **VSI_CACHE=[TRUE/FALSE]:** When using the VSI interface files can be cached in RAM by setting the configuration option `VSI_CACHE` to `TRUE`. The cache size defaults to 25 MB, but can be modified by setting the configuration option *VSI_CACHE_SIZE*. (in bytes). When enabled, this cache is used for most I/O in GDAL, including local files.
- **VSI_CACHE_SIZE=<size in bytes>:** Set the size of the VSI cache. Be wary of large values for `VSI_CACHE_SIZE` when opening VRT datasources containing many source rasters, as this is a per-file cache. Since GDAL 3.11, the value of `VSI_CACHE_SIZE` may be specified using memory units (e.g., "25 MB").

8.15.4.3 Driver management

- **GDAL_SKIP=comma-separated list:** Used by *GDALDriverManager::AutoSkipDrivers()* This option can be used to unregister one or several GDAL drivers. This can be useful when a driver tries to open a dataset that it should not recognize, or when several drivers are built-in that can open the same datasets (for example JP2MrSID, JP2ECW, JPEG2000 and JP2KAK for JPEG2000 datasets). The value of this option must be a comma delimited list of the short name of the GDAL drivers to unregister.

  This option must be set before calling *GDALAllRegister()*, or an explicit call to *GDALDriverManager::AutoSkipDrivers()* will be required.
- **OGR_SKIP=comma-separated list:** This option can be used to unregister one or several OGR drivers. This can be useful when a driver tries to open a datasource that it should not recognize, or when several drivers are built-in that can open the same datasets (for example KML, LIBKML datasources). The value of this option must be a comma delimited list of the short name of the OGR drivers to unregister.
- **GDAL_DRIVER_PATH=value:** Used by *GDALDriverManager::AutoLoadDrivers()*. This function will automatically load drivers from shared libraries. It searches the "driver path" for .so (or .dll) files that start with the prefix "gdal_X.so". It then tries to load them and then tries to call a function within them called GDALRegister_X() where the 'X' is the same as the remainder of the shared library basename ('X' is case sensitive), or failing that to call GDALRegisterMe().

There are a few rules for the driver path. If the `GDAL_DRIVER_PATH` environment variable it set, it is taken to be a list of directories to search separated by colons on UNIX, or semi-colons on Windows. Otherwise the /usr/local/lib/gdalplugins directory, and (if known) the lib/gdalplugins subdirectory of the gdal home directory are searched on UNIX and $(BINDIR)gdalplugins on Windows.

Auto loading can be completely disabled by setting the `GDAL_DRIVER_PATH` config option to "disable".

This option must be set before calling *`GDALAllRegister()`*, or an explicit call to *`GDALDriverManager::AutoLoadDrivers()`* will be required.

- **GDAL_PYTHON_DRIVER_PATH=value:** A list of directories to search for `.py` files implementing GDAL drivers. Like *`GDAL_DRIVER_PATH`*, directory names should be separated by colons on Unix or semi-colons on Windows. For more information, see rfc-76.

8.15.4.4 General options

- **GDAL_DATA=<path>:** Path to directory containing various GDAL data files (EPSG CSV files, S-57 definition files, DXF header and footer files, ...). This option is read by the GDAL and OGR driver registration functions. It is used to expand EPSG codes into their description in the OSR model (WKT based).

  On some builds (Unix), the value can be hard-coded at compilation time to point to the path after installation (/usr/share/gdal/data for example). On Windows platform, this option must be generally declared.

- **GDAL_CONFIG_FILE=value:** (GDAL >= 3.3) The location of the GDAL config file (see *GDAL configuration file*).

- **CPL_TMPDIR=<dirname>:** By default, temporary files are written into current working directory. Sometimes this is not optimal and it would be better to write temporary files on bigger or faster drives (SSD).

- **GDAL_RASTERIO_RESAMPLING=[NEAR/BILINEAR/CUBIC/CUBICSPLINE/LANCZOS/AVERAGE/RMS/MODE/GAUSS]:** Defaults to `NEAR`. Sets the resampling algorithm to be used when reading from a raster into a buffer with different dimensions from the source region.

- **CPL_VSIL_ZIP_ALLOWED_EXTENSIONS=<comma-separated list>:** Add to zip FS handler default extensions array (zip, kmz, dwf, ods, xlsx) additional extensions listed in *`CPL_VSIL_ZIP_ALLOWED_EXTENSIONS`* config option.

- **CPL_VSIL_DEFLATE_CHUNK_SIZE=value:** Defaults to `1M`.

- **GDAL_DISABLE_CPLLOCALEC=[YES/NO]:** Defaults to `NO`. If set to YES (default is NO) this option will disable the normal behavior of the CPLLocaleC class which forces the numeric locale to "C" for selected chunks of code using the setlocale() call. Behavior of setlocale() in multi-threaded applications may be undependable but use of this option may result in problem formatting and interpreting numbers properly.

- **GDAL_FILENAME_IS_UTF8=[YES/NO]:** Defaults to `YES`. This option only has an effect on Windows systems (using cpl_vsil_win32.cpp). If set to "NO" then filenames passed to functions like *`VSIFOpenL()`* will be passed on directly to CreateFile() instead of being converted from UTF-8 to wchar_t and passed to CreateFileW(). This effectively restores the pre-GDAL1.8 behavior for handling filenames on Windows and might be appropriate for applications that treat filenames as being in the local encoding.

- **GDAL_MAX_BAND_COUNT=<integer>:** Defaults to `65536`. Defines the maximum number of bands to read from a single dataset.

- **GDAL_XML_VALIDATION=[YES/NO]:** Defaults to `YES`. Determines whether XML content should be validated against an XSD, with non-conformities reported as warnings.

- **GDAL_GEOREF_SOURCES=value:** Determines the order in which potential georeferencing sources are scanned. Value should be a comma-separated list of sources in order of decreasing priority. The set of sources recognized by this option is driver-specific.

- **GDAL_OVR_PROPAGATE_NODATA=[YES/NO]:** Defaults to `NO`. When computing the value of an overview pixel, determines whether a single NODATA value should cause the overview pixel to be set to NODATA (`YES`), or whether the NODATA values should be simply ignored (`NO`). This configuration option is not supported for all resampling algorithms/data types.
- **USE_RRD=[YES/NO]:** Defaults to `NO`. Used by gdaldefaultoverviews.cpp Can be set to YES to use Erdas Imagine format (.aux) as overview format. See **gdaladdo** documentation.
- **PYTHONSO=value:** Location of Python shared library file, e.g. `pythonX.Y[...].so/.dll`.
- **CPL_ENABLE_PATH_TRAVERSAL_DETECTION=[YES/NO]:** (GDAL >= 3.12) Defaults to `YES`. Whether *`CPLHasPathTraversal()`* must detect `../` or `..\\` patterns in file paths that could cause path traversal vulnerabilities.

8.15.4.5 Vector related options

- **OGR_ARC_STEPSIZE=<degrees>:** Defaults to `4`. Used by *`OGR_G_CreateFromGML()`* (for gml:Arc and gml:Circle) and *`OGRGeometryFactory::approximateArcAngles()`* to stroke arc to linestrings. The approximation of arcs as linestrings is done by splitting the arcs into subarcs of no more than the angle specified by this option.
- **OGR_ARC_MAX_GAP=value:** Defaults to `0`. Arcs will be approximated while enforcing a maximum distance between adjacent points on the interpolated curve. Setting this option to 0 (the default) means no maximum distance applies.
- **OGR_STROKE_CURVE=[TRUE/FALSE]:** Defaults to `FALSE`. Controls whether curved geometries should be approximated by linear geometries.
- **OGR_ORGANIZE_POLYGONS=[DEFAULT/SKIP/ONLY_CCW/CCW_INNER_JUST_AFTER_CW_OUTER]:** Defines the method used to classify polygon rings as holes or shells. Although one of the options is named `DEFAULT`, some drivers may default to a different method to reduce processing by taking advantage of a format's constraints. The following methods are available, in order of decreasing expected runtime:
  - `DEFAULT`: perform a full analysis of the topological relationships between all rings, classifying them as shells or holes and associating them according to the OGC Simple Features convention. If the topological analysis determines that a valid geometry cannot be constructed, the result will be the same as with *`OGR_ORGANIZE_POLYGONS=SKIP`*.
  - `ONLY_CCW`: assume that rings with clockwise orientation represent shells and rings with counterclockwise orientation represent holes. Perform a limited topological analysis to determine which shell contains each hole. The Shapefile driver defaults to this method.
  - `CCW_INNER_JUST_AFTER_CW_OUTER`: assume that rings with clockwise orientation represent shells and rings with counterclockwise orientation represent holes and immediately follow the outer ring with which they are associated.
  - `SKIP`: avoid attempting to classify rings as shells or holes. A single geometry (Polygon/MultiPolygon/CurvePolygon/MultiSurface) will be returned with all polygons as top-level polygons. If non-polygonal elements are present, a GeometryCollection will be returned.
- **OGR_SQL_LIKE_AS_ILIKE=[YES/NO]:** (GDAL >= 3.1) Defaults to `NO`. If `YES`, the LIKE operator in the OGR SQL dialect will be case-insensitive (ILIKE), as was the case for GDAL versions prior to 3.1.
- **OGR_FORCE_ASCII=[YES/NO]:** Defaults to `YES`. Used by `OGRGetXML_UTF8_EscapedString()` function and by GPX, KML, GeoRSS and GML drivers. Those XML based drivers should write UTF8 content. If they are provided with non UTF8 content, they will replace each non-ASCII character by '?' when OGR_FORCE_ASCII=YES.

  Set to NO to preserve the content, but beware that the resulting XML file will not be valid and will require manual edition of the encoding in the XML header.

- **OGR_APPLY_GEOM_SET_PRECISION=[YES/NO]:** (GDAL >= 3.9) Defaults to NO. By default, when a geometry coordinate precision is set on a geometry field definition and a driver honors the GDAL_DCAP_HONOR_GEOM_COORDINATE_PRECISION capability, geometries passed to *OGRLayer::CreateFeature()* and *OGRLayer::SetFeature()* are assumed to be compatible with the coordinate precision. That is they are assumed to be valid once their coordinates are rounded to it. If it might not be the case, set this configuration option to YES before calling CreateFeature() or SetFeature() to force *OGRGeometry::SetPrecision()* to be called on the passed geometries.

8.15.4.6 Networking options

- **CPL_VSIL_CURL_ALLOWED_EXTENSIONS=<comma-separated list>:** Consider that only the files whose extension ends up with one that is listed in *CPL_VSIL_CURL_ALLOWED_EXTENSIONS* exist on the server. This can speed up dramatically open experience, in case the server cannot return a file list. For example:

```
gdalinfo --config CPL_VSIL_CURL_ALLOWED_EXTENSIONS ".tif" /vsicurl/http://
↪igskmncngs506.cr.usgs.gov/gmted/Global_tiles_GMTED/075darcsec/bln/W030/30N030W_
↪20101117_gmted_bln075.tif
```

- **CPL_VSIL_CURL_CACHE_SIZE=<bytes>:** Defaults to 16 MB. Size of global least-recently-used (LRU) cache shared among all downloaded content. Value is assumed to represent bytes unless memory units are specified (since GDAL 3.11).
- **CPL_VSIL_CURL_USE_HEAD=[YES/NO]:** Defaults to YES. Controls whether to use a HEAD request when opening a remote URL.
- **CPL_VSIL_CURL_USE_S3_REDIRECT=[YES/NO]:** Defaults to YES. Try to query quietly redirected URLs to Amazon S3 signed URLs during their validity period, so as to minimize round-trips.
- **CPL_VSIL_CURL_AUTHORIZATION_HEADER_ALLOWED_IF_REDIRECT=[YES/NO/IF_SAME_HOST]:** (GDAL >= 3.10) Defaults to IF_SAME_HOST. Determines if the HTTP Authorization header must be forwarded when redirections are followed:
  - NO to always disable forwarding of Authorization header
  - YES to always enable forwarding of Authorization header (was the default value prior to GDAL 3.10)
  - IF_SAME_HOST to enable forwarding of Authorization header only if the redirection is to the same host.
- **CPL_VSIL_USE_TEMP_FILE_FOR_RANDOM_WRITE=[YES/NO]:** Use a local temporary file to support random writes in certain virtual file systems. The temporary file will be located in *CPL_TMPDIR*.
- **CURL_CA_BUNDLE=value:** Set the path to the Certification Authority (CA) bundle file.
- **SSL_CERT_FILE=value:**
- **CPL_VSIL_CURL_CHUNK_SIZE=<bytes>:** Value is assumed to represent bytes unless memory units are specified (since GDAL 3.11).
- **GDAL_INGESTED_BYTES_AT_OPEN=value:** Sets the number of bytes read in one GET call at file opening.
- **CPL_VSIL_CURL_NON_CACHED=<colon-separated list>:** A global LRU cache of 16 MB shared among all downloaded content is enabled by default, and content in it may be reused after a file handle has been closed and reopened. The *CPL_VSIL_CURL_NON_CACHED* configuration option can be set to values like /vsis3/bucket/foo.tif:/vsis3/another_bucket/some_directory, so that at file handle closing, all cached content related to the mentioned file(s) is no longer cached. This can help when dealing with resources that can be modified during execution of GDAL-related code.
- **GDAL_HTTP_PATH_VERBATIM=[YES/NO]:** (GDAL >= 3.12) Defaults to NO. When set to YES, sequences of /../ or /./ that may exist in the URL's path part are kept unchanged. Otherwise, by default, they are squashed, according to RFC 3986 section 5.2.4.

- **GDAL_HTTP_HEADER_FILE=<filename>:** Filename of a text file with "key: value" HTTP headers. The content of the file is not cached, and thus it is read again before issuing each HTTP request.
- **GDAL_HTTP_CONNECTTIMEOUT=<seconds>:** Maximum delay for connection to be established before being aborted.
- **GDAL_HTTP_COOKIE=value:** Cookie(s) to send. See https://curl.se/libcurl/c/CURLOPT_COOKIE.html
- **GDAL_HTTP_COOKIEFILE=value:** File name to read cookies from. See https://curl.se/libcurl/c/CURLOPT_COOKIEFILE.html
- **GDAL_HTTP_COOKIEJAR=value:** File to which cookies should be written. See https://curl.se/libcurl/c/CURLOPT_COOKIEJAR.html
- **GDAL_HTTP_NETRC=[YES/NO]:** Defaults to `YES`. Controls if an available `.netrc` file is used.
- **GDAL_HTTP_NETRC_FILE=<filename>:** (GDAL >= 3.7.0) Sets the location of a `.netrc` file.
- **GDAL_HTTP_LOW_SPEED_LIMIT=<bytes/s>:** Defaults to `0`. Sets the transfer speed, averaged over *GDAL_HTTP_LOW_SPEED_TIME*, below which a request should be canceled.
- **GDAL_HTTP_LOW_SPEED_TIME=<seconds>:** Defaults to `0`. Sets the time window over which *GDAL_HTTP_LOW_SPEED_LIMIT* should be evaluated.
- **GDAL_HTTP_SSL_VERIFYSTATUS=[YES/NO]:** Defaults to `NO`. Whether to verify the status of SSL certificates. See https://curl.se/libcurl/c/CURLOPT_SSL_VERIFYSTATUS.html
- **GDAL_HTTP_USE_CAPI_STORE=[YES/NO]:** Defaults to `NO`. (Windows only). Whether to use certificates from the Windows certificate store.
- **GDAL_HTTP_HEADERS=value:** (GDAL >= 3.6) Specifies headers as a comma separated list of key: value pairs. If a comma or a double-quote character is needed in the value, then the key: value pair must be enclosed in double-quote characters. In that situation, backslash and double quote character must be backslash-escaped. e.g GDAL_HTTP_HEADERS=Foo: Bar,"Baz: escaped backslash \\, escaped double-quote \", end of value",Another: Header
- **GDAL_HTTP_MAX_RETRY=value:** Defaults to `0`. Set the number of HTTP attempts, when a retry is allowed. (cf *GDAL_HTTP_RETRY_CODES* for conditions where a retry is attempted.) The default value is 0, meaning no retry.
- **GDAL_HTTP_RETRY_DELAY=<seconds>:** Defaults to `30`. Set the delay between HTTP attempts.
- **GDAL_HTTP_RETRY_CODES=ALL or comma-separated list of codes:** (GDAL >= 3.10) Specify which HTTP error codes should trigger a retry attempt. Valid values are `ALL` or a comma-separated list of HTTP codes. By default, 429, 500, 502, 503 or 504 HTTP errors are considered, as well as other situations with a particular HTTP or Curl error message.
- **GDAL_HTTP_TCP_KEEPALIVE=[YES/NO]:** (GDAL >= 3.6) Defaults to `NO`. Sets whether to enable TCP keep-alive.
- **GDAL_HTTP_TCP_KEEPIDLE=<seconds>:** (GDAL >= 3.6) Defaults to `60`. Keep-alive idle time. Only taken into account if *GDAL_HTTP_TCP_KEEPALIVE=YES*.
- **GDAL_HTTP_TCP_KEEPINTVL=<seconds>:** (GDAL >= 3.6) Defaults to `60`. Interval time between keep-alive probes. Only taken into account if *GDAL_HTTP_TCP_KEEPALIVE=YES*.
- **GDAL_HTTP_SSLCERT=<filename>:** (GDAL >= 3.7) Filename of the the SSL client certificate. See https://curl.se/libcurl/c/CURLOPT_SSLCERT.html
- **GDAL_HTTP_SSLCERTTYPE=[PEM/DER]:** (GDAL >= 3.7) Format of the SSL certificate. see https://curl.se/libcurl/c/CURLOPT_SSLCERTTYPE.html
- **GDAL_HTTP_SSLKEY=<filename>:** (GDAL >= 3.7) Private key file for TLS and SSL client certificate. see https://curl.se/libcurl/c/CURLOPT_SSLKEY.html

- **GDAL_HTTP_KEYPASSWD=value:** (GDAL >= 3.7) Passphrase to private key. See https://curl.se/libcurl/c/CURLOPT_KEYPASSWD.html
- **GDAL_HTTP_VERSION=[1.0/1.1/2/2TLS/2PRIOR_KNOWLEDGE]:** Specifies which HTTP version to use. Will default to 1.1 generally (except on some controlled environments, like Google Compute Engine VMs, where 2TLS will be the default). Support for HTTP/2 requires curl 7.33 or later, built against nghttp2. "2TLS" means that HTTP/2 will be attempted for HTTPS connections only. Whereas "2" means that HTTP/2 will be attempted for HTTP or HTTPS. "2PRIOR_KNOWLEDGE" means that the server will be assumed to support HTTP/2. The interest of enabling HTTP/2 is the use of HTTP/2 multiplexing when reading GeoTIFFs stored on /vsicurl/ and related virtual file systems.
- **GDAL_HTTP_MULTIPLEX=[YES/NO]:** Defaults to YES. Only applies on a HTTP/2 connection. If set to YES, HTTP/2 multiplexing can be used to download multiple ranges in parallel, during ReadMultiRange() requests that can be emitted by the GeoTIFF driver.
- **GDAL_HTTP_MULTIRANGE=[SINGLE_GET/SERIAL/YES]:** Defaults to `YES`. Controls how ReadMultiRange() requests emitted by the GeoTIFF driver are satisfied. SINGLE_GET means that several ranges will be expressed in the Range header of a single GET requests, which is not supported by a majority of servers (including AWS S3 or Google GCS). SERIAL means that each range will be requested sequentially. YES means that each range will be requested in parallel, using HTTP/2 multiplexing or several HTTP connections.
- **GDAL_HTTP_MERGE_CONSECUTIVE_RANGES=[YES/NO]:** Defaults to `YES`. Only applies when *GDAL_HTTP_MULTIRANGE* is YES. Defines if ranges of a single ReadMultiRange() request that are consecutive should be merged into a single request.
- **GDAL_HTTP_AUTH=[BASIC/NTLM/NEGOTIATE/ANY/ANYSAFE/BEARER]:** Set value to tell libcurl which authentication method(s) you want it to use. See http://curl.haxx.se/libcurl/c/curl_easy_setopt.html#CURLOPTHTTPAUTH for more information.
- **GDAL_HTTP_USERPWD=value:** The HTTP user and password to use for the connection. Must be in the form of [user name]:[password]. Use *GDAL_HTTP_AUTH* to decide the authentication method. When using NTLM, you can set the domain by prepending it to the user name and separating the domain and name with a forward (/) or backward slash (). Like this: "domain/user:password" or "domainuser:password". Some HTTP servers (on Windows) support this style even for Basic authentication.
- **GDAL_GSSAPI_DELEGATION=[NONE/POLICY/ALWAYS]:** (GDAL >= 3.3) Set allowed GSS-API delegation. Relevant only with *GDAL_HTTP_AUTH=NEGOTIATE*.
- **GDAL_HTTP_BEARER=value:** (GDAL >= 3.9) Set HTTP OAuth 2.0 Bearer Access Token to use for the connection. Must be used with *GDAL_HTTP_AUTH=BEARER*.
- **GDAL_HTTP_PROXY=value:** Set HTTP proxy to use. The parameter should be the host name or dotted IP address. To specify port number in this string, append :[port] to the end of the host name. The proxy string may be prefixed with [protocol]: since any such prefix will be ignored. The proxy's port number may optionally be specified with the separate option. If not specified, libcurl will default to using port 1080 for proxies. GDAL respects the environment variables http_proxy, ftp_proxy, all_proxy etc, if any of those are set. GDAL_HTTP_PROXY option does however override any possibly set environment variables.
- **GDAL_HTTPS_PROXY=value:** Set HTTPS proxy to use. See *GDAL_HTTP_PROXY*.
- **GDAL_HTTP_PROXYUSERPWD=value:** The HTTP user and password to use for the connection to the HTTP proxy. Must be in the form of [user name]:[password].
- **GDAL_PROXY_AUTH=[BASIC/NTLM/NEGOTIATE/DIGEST/ANY/ANYSAFE]:** Set value to tell libcurl which authentication method(s) you want it to use for your proxy authentication. See http://curl.haxx.se/libcurl/c/curl_easy_setopt.html#CURLOPTPROXYAUTH for more information.
- **GDAL_HTTP_MAX_CACHED_CONNECTIONS=value:** (GDAL >= 3.11) Maximum amount of connections that libcurl may keep alive in its connection cache after use. Cf https://curl.se/libcurl/c/CURLMOPT_MAXCONNECTS.html

- **GDAL_HTTP_MAX_TOTAL_CONNECTIONS=value:** (GDAL >= 3.11) Maximum number of simultaneously open connections in total. Cf https://curl.se/libcurl/c/CURLMOPT_MAX_TOTAL_CONNECTIONS.html
- **CPL_CURL_GZIP=[YES/NO]:** Sets the contents of the Accept-Encoding header sent in a HTTP request to gzip, and enables decoding of a response when a Content-Encoding: header
- **GDAL_HTTP_TIMEOUT=value:** Set HTTP timeout value, where value is in seconds
- **GDAL_HTTP_USERAGENT=value:** This string will be used to set the `User-Agent` header in the HTTP request sent to the remote server. Defaults to "GDAL/x.y.z" where x.y.z is the GDAL build version.
- **GDAL_HTTP_UNSAFESSL=[YES/NO]:** Defaults to `NO`. Set to "YES" to get the curl library to skip SSL host / certificate verification.

8.15.4.7 Persistent Auxiliary Metadata (PAM) options

- **GDAL_PAM_ENABLED=[YES/NO]:** PAM support can be enabled (resp. disabled) in GDAL by setting the *GDAL_PAM_ENABLED* configuration option (via CPLSetConfigOption(), or the environment) to the value of YES (resp. NO). Note: The default value is build dependent and defaults to YES in Windows and Unix builds. See `GDALPamDataset` for more information. Note that setting this option to OFF may have subtle/silent side-effects on various drivers that rely on PAM functionality.
- **GDAL_PAM_PROXY_DIR=value:** Directory to which `.aux.xml` files will be written when accessing files from a location where the user does not have write permissions. Has no effect when accessing files from locations where the user does have write permissions. Must be set before the first access to PAM.

8.15.4.8 PROJ options

- **CENTER_LONG=value:**
- **CHECK_WITH_INVERT_PROJ=value:** Defaults to `NO`. Used by ogrct.cpp and gdalwarp_lib.cpp. This option can be used to control the behavior of **gdalwarp** when warping global datasets or when transforming from/to polar projections, which causes coordinate discontinuities. See http://trac.osgeo.org/gdal/ticket/2305.

  The background is that PROJ does not guarantee that converting from `src_srs` to `dst_srs` and then from `dst_srs` to `src_srs` will yield the initial coordinates. This can lead to errors in the computation of the target bounding box of **gdalwarp**, or to visual artifacts.

  If *CHECK_WITH_INVERT_PROJ* option is not set, **gdalwarp** will check that the computed coordinates of the edges of the target image are in the validity area of the target projection. If they are not, it will retry computing them by setting *CHECK_WITH_INVERT_PROJ=TRUE* that forces ogrct.cpp to check the consistency of each requested projection result with the inverse projection.

  If set to `NO`, **gdalwarp** will not attempt to use the inverse projection.
- **THRESHOLD=value:** Defaults to `0.1` for geographic SRS, 10000 otherwise. Used by ogrct.cpp. Used in combination with *CHECK_WITH_INVERT_PROJ=TRUE*. Defines the acceptable threshold used to check if the round-trip from `src_srs` to `dst_srs` and from `dst_srs` to `srs_srs` yields the initial coordinates. The round-trip transformation will be considered successful if the `x` and `y` values are both within *THRESHOLD* of the original values. The value must be expressed in the units of the source SRS (typically degrees for a geographic SRS, meters for a projected SRS).
- **OGR_ENABLE_PARTIAL_REPROJECTION=[YES/NO]:** Defaults to `NO`. Used by `OGRLineString::transform()`. Can be set to YES to remove points that cannot be reprojected. This can for example help reproject lines that have an extremity at a pole, when the reprojection does not support coordinates at poles.
- **OGR_CT_USE_SRS_COORDINATE_EPOCH=[YES/NO]:** If `NO`, disables the coordinate epoch associated with the target or source CRS when transforming between a static and dynamic CRS.

- **OSR_ADD_TOWGS84_ON_EXPORT_TO_WKT1=[YES/NO]:** (GDAL >= 3.0.3) Defaults to `NO`. Determines whether a `TOWGS84` node should be automatically added when exporting a CRS to the GDAL flavor of WKT1.
- **OSR_ADD_TOWGS84_ON_EXPORT_TO_PROJ4=[YES/NO]:** (GDAL >= 3.0.3) Defaults to `YES`. Determines whether a `+towgs84` parameter should be automatically added when exporting a CRS as a legacy PROJ.4 string.
- **OSR_ADD_TOWGS84_ON_IMPORT_FROM_EPSG=[YES/NO]:** (GDAL >= 3.0.3) Defaults to `NO`. Determines whether to automatically add a 3-parameter or 7-parameter Helmert transformation to WGS84 when there is exactly one such method available for the CRS.
- **OSR_DEFAULT_AXIS_MAPPING_STRATEGY=[TRADITIONAL_GIS_ORDER/AUTHORITY_COMPLIANT]:** (GDAL >= 3.5) Defaults to `AUTHORITY_COMPLIANT`. Determines whether to honor the declared axis mapping of a CRS or override it with the traditional GIS ordering (x = longitude, y = latitude).
- **OSR_STRIP_TOWGS84=[YES/NO]:** (GDAL >= 3.1) Defaults to `YES`. Determines whether to remove TOWGS84 information if the CRS has a known horizontal datum.
- **OSR_USE_NON_DEPRECATED=[YES/NO]:** Defaults to `YES`. Determines whether to substitute a replacement for deprecated EPSG codes.
- **OSR_WKT_FORMAT=[SFSQL/WKT1_SIMPLE/WKT1/WKT1_GDAL/WKT1_ESRI/WKT2_2015/WKT2_2018/WKT2/DEFAULT]:** Defaults to `DEFAULT`. Sets the format for writing a CRS to WKT.

8.15.5 List of configuration options and where they are documented

- `AAIGRID_DATATYPE`: *AAIGrid -- Arc/Info ASCII Grid*
- `AMIGOCLOUD_API_KEY`: *AmigoCloud*
- `AMIGOCLOUD_API_URL`: *AmigoCloud*
- `ATTRIBUTES_SKIP`: *GeoJSON*
- `AWS_ACCESS_KEY_ID`: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `AWS_CONFIG_FILE`: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `AWS_DEFAULT_PROFILE`: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `AWS_DEFAULT_REGION`: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `AWS_HTTPS`: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `AWS_NO_SIGN_REQUEST`: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `AWS_PROFILE`: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `AWS_REGION`: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `AWS_REQUEST_PAYER`: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*

- AWS_ROLE_ARN: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- AWS_S3SESSION_TOKEN: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- AWS_S3_ENDPOINT: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- AWS_SECRET_ACCESS_KEY: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- AWS_SESSION_TOKEN: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- AWS_VIRTUAL_HOSTING: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- AWS_WEB_IDENTITY_TOKEN_FILE: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- AZURE_IMDS_CLIENT_ID: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- AZURE_IMDS_MSI_RES_ID: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- AZURE_IMDS_OBJECT_ID: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- AZURE_NO_SIGN_REQUEST: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- AZURE_STORAGE_ACCESS_KEY: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- AZURE_STORAGE_ACCESS_TOKEN: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- AZURE_STORAGE_ACCOUNT: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- AZURE_STORAGE_CONNECTION_STRING: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- AZURE_STORAGE_SAS_TOKEN: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- BIGTIFF_OVERVIEW:
  - *GTiff -- GeoTIFF File Format*
  - *gdaladdo*
- CARTO_API_KEY: *Carto*
- CARTO_API_URL: *Carto*
- CARTO_HTTPS: *Carto*
- CARTO_MAX_CHUNK_SIZE: *Carto*
- CARTO_PAGE_SIZE: *Carto*
- CENTER_LONG: *Configuration options*

- CHECK_DISK_FREE_SPACE: *GTiff -- GeoTIFF File Format*
- CHECK_WITH_INVERT_PROJ: *Configuration options*
- COMPRESS_GEOM: *SQLite / Spatialite RDBMS*
- COMPRESS_OVERVIEW:
  - *GTiff -- GeoTIFF File Format*
  - *gdaladdo*
- CPL_ACCUM_ERROR_MSG: *Configuration options*
- CPL_AWS_CREDENTIALS_FILE: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- CPL_CURL_GZIP:
  - *WFS - OGC WFS service*
  - *Configuration options*
- CPL_CURL_VERBOSE:
  - *Configuration options*
  - *GDAL CLI Debug and Logging Output*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- CPL_DEBUG:
  - *Error handling in GDAL*
  - *VRT -- GDAL Virtual Format*
  - *GML - Geography Markup Language*
  - *Configuration options*
- CPL_ENABLE_PATH_TRAVERSAL_DETECTION: *Configuration options*
- CPL_GS_CREDENTIALS_FILE: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- CPL_LOG:
  - *Configuration options*
  - *GDAL CLI Debug and Logging Output*
- CPL_LOG_ERRORS:
  - *Configuration options*
  - *GDAL CLI Debug and Logging Output*
- CPL_MACHINE_IS_GCE:
  - *EEDAI - Google Earth Engine Data API Image*
  - *Google Earth Engine Data API*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- CPL_MAX_ERROR_REPORTS: *Configuration options*
- CPL_SOZIP_ENABLED: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*

- CPL_SOZIP_MIN_FILE_SIZE: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- CPL_TIMESTAMP:
  - *Configuration options*
  - *GDAL CLI Debug and Logging Output*
- CPL_TMPDIR:
  - *COG -- Cloud Optimized GeoTIFF generator*
  - *PDF -- Geospatial PDF*
  - *WMS -- Web Map Services*
  - *WMTS -- OGC Web Map Tile Service*
  - *OSM - OpenStreetMap XML and PBF*
  - *<literal>gdal external</literal>*
  - *<literal>gdal raster materialize</literal>*
  - *<literal>gdal vector materialize</literal>*
  - *Common options for vector programs*
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- CPL_VSIL_CURL_ALLOWED_EXTENSIONS: *Configuration options*
- CPL_VSIL_CURL_AUTHORIZATION_HEADER_ALLOWED_IF_REDIRECT:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- CPL_VSIL_CURL_CACHE_SIZE:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- CPL_VSIL_CURL_CHUNK_SIZE:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- CPL_VSIL_CURL_IGNORE_GLACIER_STORAGE: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- CPL_VSIL_CURL_IGNORE_STORAGE_CLASSES: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- CPL_VSIL_CURL_NON_CACHED:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- CPL_VSIL_CURL_USE_HEAD:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*

- *CPL_VSIL_CURL_USE_S3_REDIRECT*:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- *CPL_VSIL_DEFLATE_CHUNK_SIZE*:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- *CPL_VSIL_GZIP_WRITE_PROPERTIES*: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- *CPL_VSIL_USE_TEMP_FILE_FOR_RANDOM_WRITE*:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- *CPL_VSIL_ZIP_ALLOWED_EXTENSIONS*: *Configuration options*
- *CPL_VSIS3_CREATE_DIR_OBJECT*: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- *CPL_VSIS3_USE_BASE_RMDIR_RECURSIVE*: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- *CPL_VSISTDIN_BUFFER_LIMIT*: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- *CURL_CA_BUNDLE*:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- *DECIMAL_PRECISION*: *AAIGrid -- Arc/Info ASCII Grid*
- *DTED_APPLY_PIXEL_IS_POINT*: *DTED -- Military Elevation Data*
- *DTED_ASSUME_CONFORMANT*: *DTED -- Military Elevation Data*
- *DTED_VERIFY_CHECKSUM*: *DTED -- Military Elevation Data*
- *DWG_ALL_ATTRIBUTES*: *AutoCAD DWG*
- *DWG_ATTRIBUTES*: *AutoCAD DWG*
- *DWG_CLOSED_LINE_AS_POLYGON*: *AutoCAD DWG*
- *DWG_INLINE_BLOCKS*: *AutoCAD DWG*
- *DXF_3D_EXTENSIBLE_MODE*: *AutoCAD DXF*
- *DXF_CLOSED_LINE_AS_POLYGON*: *AutoCAD DXF*
- *DXF_ENCODING*: *AutoCAD DXF*
- *DXF_FEATURE_LIMIT_PER_BLOCK*: *AutoCAD DXF*
- *DXF_HATCH_TOLERANCE*: *AutoCAD DXF*
- *DXF_INCLUDE_RAW_CODE_VALUES*: *AutoCAD DXF*
- *DXF_INLINE_BLOCKS*: *AutoCAD DXF*
- *DXF_MERGE_BLOCK_GEOMETRIES*: *AutoCAD DXF*

- DXF_TRANSLATE_ESCAPE_SEQUENCES: *AutoCAD DXF*
- DXF_WRITE_HATCH: *AutoCAD DXF*
- ECWP_BLOCKING_TIME_MS: *ECW -- Enhanced Compressed Wavelets (.ecw)*
- ECWP_CACHE_LOCATION: *ECW -- Enhanced Compressed Wavelets (.ecw)*
- ECWP_CACHE_SIZE_MB: *ECW -- Enhanced Compressed Wavelets (.ecw)*
- ECWP_REFRESH_TIME_MS: *ECW -- Enhanced Compressed Wavelets (.ecw)*
- ECW_ALWAYS_UPWARD: *ECW -- Enhanced Compressed Wavelets (.ecw)*
- ECW_CACHE_MAXMEM:
  - *ECW -- Enhanced Compressed Wavelets (.ecw)*
  - *JP2ECW -- ERDAS JPEG2000 (.jp2)*
- ECW_CACHE_MAXOPEN: *ECW -- Enhanced Compressed Wavelets (.ecw)*
- ECW_FORCE_FILE_REOPEN: *ECW -- Enhanced Compressed Wavelets (.ecw)*
- ECW_RESILIENT_DECODING: *ECW -- Enhanced Compressed Wavelets (.ecw)*
- ECW_TEXTURE_DITHER:
  - *ECW -- Enhanced Compressed Wavelets (.ecw)*
  - *JP2ECW -- ERDAS JPEG2000 (.jp2)*
- EEDA_BEARER:
  - *EEDAI - Google Earth Engine Data API Image*
  - *Google Earth Engine Data API*
- EEDA_BEARER_FILE:
  - *EEDAI - Google Earth Engine Data API Image*
  - *Google Earth Engine Data API*
- EEDA_CLIENT_EMAIL:
  - *EEDAI - Google Earth Engine Data API Image*
  - *Google Earth Engine Data API*
- EEDA_PAGE_SIZE: *Google Earth Engine Data API*
- EEDA_PRIVATE_KEY:
  - *EEDAI - Google Earth Engine Data API Image*
  - *Google Earth Engine Data API*
- EEDA_PRIVATE_KEY_FILE:
  - *EEDAI - Google Earth Engine Data API Image*
  - *Google Earth Engine Data API*
- ESRI_XML_PAM: *GTiff -- GeoTIFF File Format*
- ES_BULK: *Elasticsearch: Geographically Encoded Objects for Elasticsearch*
- ES_META: *Elasticsearch: Geographically Encoded Objects for Elasticsearch*
- ES_OVERWRITE: *Elasticsearch: Geographically Encoded Objects for Elasticsearch*

- ES_WRITEMAP: *Elasticsearch: Geographically Encoded Objects for Elasticsearch*
- FORCE_CELLSIZE: *AAIGrid -- Arc/Info ASCII Grid*
- GDAL_BAND_BLOCK_CACHE:
  - *Configuration options*
- GDAL_CACHEMAX:
  - *General API*
  - *SENTINEL2 -- Sentinel-2 Products*
  - *gdalwarp*
  - *Common options for raster programs*
  - *Configuration options*
- GDAL_CONFIG_FILE: *Configuration options*
- GDAL_DAAS_ACCESS_TOKEN: *DAAS (Airbus DS Intelligence Data As A Service driver)*
- GDAL_DAAS_API_KEY: *DAAS (Airbus DS Intelligence Data As A Service driver)*
- GDAL_DAAS_CLIENT_ID: *DAAS (Airbus DS Intelligence Data As A Service driver)*
- GDAL_DAAS_X_FORWARDED_USER: *DAAS (Airbus DS Intelligence Data As A Service driver)*
- GDAL_DATA:
  - *GRIB -- WMO General Regularly-distributed Information in Binary form*
  - *Common options for raster programs*
  - *Configuration options*
- GDAL_DEFAULT_WMS_CACHE_PATH:
  - *WMS -- Web Map Services*
  - *WMTS -- OGC Web Map Tile Service*
- GDAL_DISABLE_CPLLOCALEC: *Configuration options*
- GDAL_DISABLE_READDIR_ON_OPEN: *Configuration options*
- GDAL_DRIVER_PATH:
  - *Configuration options*
- GDAL_DTED_SINGLE_BLOCK: *DTED -- Military Elevation Data*
- GDAL_ENABLE_TIFF_SPLIT: *GTiff -- GeoTIFF File Format*
- GDAL_ENABLE_WMS_CACHE:
  - *WMS -- Web Map Services*
  - *WMTS -- OGC Web Map Tile Service*
- GDAL_ERROR_ON_LIBJPEG_WARNING: *JPEG -- JPEG JFIF File Format*
- GDAL_EXPRTK_ENABLE_LOOPS: *VRT processed dataset*
- GDAL_EXPRTK_MAX_EXPRESSION_LENGTH: *VRT processed dataset*
- GDAL_EXPRTK_MAX_VECTOR_LENGTH: *VRT processed dataset*
- GDAL_EXPRTK_TIMEOUT_SECONDS: *VRT processed dataset*

- GDAL_FILENAME_IS_UTF8: *Configuration options*
- GDAL_FORCE_CACHING: *Configuration options*
- GDAL_GEOREF_SOURCES:
  - *GTiff -- GeoTIFF File Format*
  - *JP2ECW -- ERDAS JPEG2000 (.jp2)*
  - *JP2Grok -- JPEG2000 driver based on Grok library*
  - *JP2KAK -- JPEG 2000 (based on Kakadu SDK)*
  - *JP2MrSID -- JPEG2000 via MrSID SDK*
  - *JP2OpenJPEG -- JPEG2000 driver based on OpenJPEG library*
  - *Configuration options*
- GDAL_GSSAPI_DELEGATION: *Configuration options*
- GDAL_HFA_OVR_BLOCKSIZE: *HFA -- Erdas Imagine .img*
- GDAL_HTTPS_PROXY:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- GDAL_HTTP_AUTH: *Configuration options*
- GDAL_HTTP_BEARER: *Configuration options*
- GDAL_HTTP_CONNECTTIMEOUT: *Configuration options*
- GDAL_HTTP_COOKIE: *Configuration options*
- GDAL_HTTP_COOKIEFILE: *Configuration options*
- GDAL_HTTP_COOKIEJAR: *Configuration options*
- GDAL_HTTP_HEADERS:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- GDAL_HTTP_HEADER_FILE:
  - *WMTS -- OGC Web Map Tile Service*
  - *WFS - OGC WFS service*
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- GDAL_HTTP_KEYPASSWD:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- GDAL_HTTP_LOW_SPEED_LIMIT: *Configuration options*
- GDAL_HTTP_LOW_SPEED_TIME: *Configuration options*
- GDAL_HTTP_MAX_CACHED_CONNECTIONS:
  - *Configuration options*

  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `GDAL_HTTP_MAX_RETRY`:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `GDAL_HTTP_MAX_TOTAL_CONNECTIONS`:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `GDAL_HTTP_MERGE_CONSECUTIVE_RANGES`: *Configuration options*
- `GDAL_HTTP_MULTIPLEX`: *Configuration options*
- `GDAL_HTTP_MULTIRANGE`: *Configuration options*
- `GDAL_HTTP_NETRC`: *Configuration options*
- `GDAL_HTTP_NETRC_FILE`: *Configuration options*
- `GDAL_HTTP_PATH_VERBATIM`:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `GDAL_HTTP_PROXY`:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `GDAL_HTTP_PROXYUSERPWD`:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `GDAL_HTTP_RETRY_CODES`:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `GDAL_HTTP_RETRY_DELAY`:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `GDAL_HTTP_SSLCERT`:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `GDAL_HTTP_SSLCERTTYPE`:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `GDAL_HTTP_SSLKEY`:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*

- GDAL_HTTP_SSL_VERIFYSTATUS: *Configuration options*
- GDAL_HTTP_TCP_KEEPALIVE:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- GDAL_HTTP_TCP_KEEPIDLE:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- GDAL_HTTP_TCP_KEEPINTVL:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- GDAL_HTTP_TIMEOUT:
  - *WMS -- Web Map Services*
  - *Configuration options*
- GDAL_HTTP_UNSAFESSL: *Configuration options*
- GDAL_HTTP_USERAGENT: *Configuration options*
- GDAL_HTTP_USERPWD: *Configuration options*
- GDAL_HTTP_USE_CAPI_STORE: *Configuration options*
- GDAL_HTTP_VERSION: *Configuration options*
- GDAL_INGESTED_BYTES_AT_OPEN:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- GDAL_JPEG_TO_RGB: *JPEG -- JPEG JFIF File Format*
- GDAL_MAX_BAND_COUNT: *Configuration options*
- GDAL_MAX_CONNECTIONS:
  - *WMS -- Web Map Services*
  - *WMTS -- OGC Web Map Tile Service*
- GDAL_MAX_DATASET_POOL_RAM_USAGE:
  - *VRT -- GDAL Virtual Format*
  - *Configuration options*
- GDAL_MAX_DATASET_POOL_SIZE:
  - *GTI -- GDAL Raster Tile Index*
  - *VRT -- GDAL Virtual Format*
  - *Configuration options*
- GDAL_MEM_ENABLE_OPEN: *MEM -- In Memory datasets*
- GDAL_NAME_AND_SHAME:
  - *Configuration options*

- GDAL_NETCDF_ASSUME_LONGLAT: *NetCDF: Network Common Data Form*
- GDAL_NETCDF_BOTTOMUP: *NetCDF: Network Common Data Form*
- GDAL_NETCDF_IGNORE_XY_AXIS_NAME_CHECKS: *NetCDF: Network Common Data Form*
- GDAL_NETCDF_REPORT_EXTRA_DIM_VALUES: *NetCDF: Network Common Data Form*
- GDAL_NETCDF_VERIFY_DIMS: *NetCDF: Network Common Data Form*
- GDAL_NUM_THREADS:
  - *BASISU -- Basis Universal*
  - *GTI -- GDAL Raster Tile Index*
  - *GTiff -- GeoTIFF File Format*
  - *JP2OpenJPEG -- JPEG2000 driver based on OpenJPEG library*
  - *JPEGXL -- JPEG-XL File Format*
  - *KTX2*
  - *LIBERTIFF -- GeoTIFF File Format*
  - *MBTiles*
  - *SENTINEL2 -- Sentinel-2 Products*
  - *VRT -- GDAL Virtual Format*
  - *Zarr*
  - *MVT: Mapbox Vector Tiles*
  - *(Geo)Parquet*
  - *PMTiles*
  - *gdal_grid*
  - *<literal>gdal vector grid</literal>*
  - *<literal>gdal vsi sozip</literal>*
  - *gdaladdo*
  - *gdalwarp*
  - *sozip*
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- GDAL_OVR_PROPAGATE_NODATA: *Configuration options*
- GDAL_PAM_ENABLED: *Configuration options*
- GDAL_PAM_PROXY_DIR:
  - *Zarr*
  - *Configuration options*
- GDAL_PDF_BANDS: *PDF -- Geospatial PDF*
- GDAL_PDF_DPI: *PDF -- Geospatial PDF*
- GDAL_PDF_LAUNDER_LAYER_NAMES: *PDF -- Geospatial PDF*

- GDAL_PDF_LAYERS: *PDF -- Geospatial PDF*
- GDAL_PDF_LAYERS_OFF: *PDF -- Geospatial PDF*
- GDAL_PDF_NEATLINE: *PDF -- Geospatial PDF*
- GDAL_PDF_RENDERING_OPTIONS: *PDF -- Geospatial PDF*
- GDAL_PROXY_AUTH:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- GDAL_PYTHON_DRIVER_PATH:
  - *Configuration options*
- GDAL_RASTERIO_RESAMPLING: *Configuration options*
- GDAL_READDIR_LIMIT_ON_OPEN: *Configuration options*
- GDAL_SKIP:
  - *NetCDF: Network Common Data Form*
  - *VRT -- GDAL Virtual Format*
  - *ogrinfo*
  - *Common options for raster programs*
  - *Common options for vector programs*
  - *Configuration options*
  - *Security considerations*
- GDAL_STACTA_SKIP_MISSING_METATILE: *STACTA - Spatio-Temporal Asset Catalog Tiled Assets*
- GDAL_SWATH_SIZE: *Configuration options*
- GDAL_TIFF_INTERNAL_MASK: *GTiff -- GeoTIFF File Format*
- GDAL_TIFF_INTERNAL_MASK_TO_8BIT: *GTiff -- GeoTIFF File Format*
- GDAL_TIFF_OVR_BLOCKSIZE: *GTiff -- GeoTIFF File Format*
- GDAL_TRY_PDS3_WITH_VICAR:
  - *PDS -- Planetary Data System v3*
  - *VICAR -- VICAR*
- GDAL_VRT_ENABLE_PYTHON:
  - *VRT -- GDAL Virtual Format*
  - *Security considerations*
- GDAL_VRT_ENABLE_RAWRASTERBAND: *VRT -- GDAL Virtual Format*
- GDAL_VRT_PYTHON_TRUSTED_MODULES:
  - *VRT -- GDAL Virtual Format*
  - *Security considerations*
- GDAL_VRT_RAWRASTERBAND_ALLOWED_SOURCE: *VRT -- GDAL Virtual Format*
- GDAL_XML_VALIDATION:

  - *VRT -- OGR Virtual Format*
  - *Configuration options*
- `GEOMETRY_AS_COLLECTION`: *GeoJSON*
- `GML_ATTRIBUTES_TO_OGR_FIELDS`: *GML - Geography Markup Language*
- `GML_CONSIDER_EPSG_AS_URN`: *GML - Geography Markup Language*
- `GML_DOWNLOAD_SCHEMA`: *GML - Geography Markup Language*
- `GML_EXPOSE_FID`: *GML - Geography Markup Language*
- `GML_EXPOSE_GML_ID`: *GML - Geography Markup Language*
- `GML_FACE_HOLE_NEGATIVE`: *GML - Geography Markup Language*
- `GML_FIELDTYPES`: *GML - Geography Markup Language*
- `GML_GET_SECONDARY_GEOM`: *GML - Geography Markup Language*
- `GML_GFS_TEMPLATE`: *GML - Geography Markup Language*
- `GML_INVERT_AXIS_ORDER_IF_LAT_LONG`:
  - *CSW - OGC CSW (Catalog Service for the Web)*
  - *GML - Geography Markup Language*
- `GML_PARSER`: *GML - Geography Markup Language*
- `GML_READ_MODE`: *GML - Geography Markup Language*
- `GML_SAVE_RESOLVED_TO`: *GML - Geography Markup Language*
- `GML_SKIP_CORRUPTED_FEATURES`: *GML - Geography Markup Language*
- `GML_SKIP_RESOLVE_ELEMS`: *GML - Geography Markup Language*
- `GML_SWAP_COORDINATES`: *GML - Geography Markup Language*
- `GML_USE_SCHEMA_IMPORT`: *GML - Geography Markup Language*
- `GOOGLE_APPLICATION_CREDENTIALS`:
  - *EEDAI - Google Earth Engine Data API Image*
  - *Google Earth Engine Data API*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `GPX_ELE_AS_25D`: *GPX - GPS Exchange Format*
- `GPX_N_MAX_LINKS`: *GPX - GPS Exchange Format*
- `GPX_SHORT_NAMES`: *GPX - GPS Exchange Format*
- `GPX_USE_EXTENSIONS`: *GPX - GPS Exchange Format*
- `GRIB_ADJUST_LONGITUDE_RANGE`: *GRIB -- WMO General Regularly-distributed Information in Binary form*
- `GRIB_NORMALIZE_UNITS`: *GRIB -- WMO General Regularly-distributed Information in Binary form*
- `GRIB_RESOURCE_DIR`: *GRIB -- WMO General Regularly-distributed Information in Binary form*
- `GS_ACCESS_KEY_ID`: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `GS_NO_SIGN_REQUEST`: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*

- GS_OAUTH2_CLIENT_EMAIL: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- GS_OAUTH2_CLIENT_ID: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- GS_OAUTH2_CLIENT_SECRET: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- GS_OAUTH2_PRIVATE_KEY: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- GS_OAUTH2_PRIVATE_KEY_FILE: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- GS_OAUTH2_REFRESH_TOKEN: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- GS_OAUTH2_SCOPE: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- GS_SECRET_ACCESS_KEY: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- GS_USER_PROJECT: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- GTIFF_DIRECT_IO: *GTiff -- GeoTIFF File Format*
- GTIFF_IGNORE_READ_ERRORS: *GTiff -- GeoTIFF File Format*
- GTIFF_LINEAR_UNITS: *GTiff -- GeoTIFF File Format*
- GTIFF_POINT_GEO_IGNORE: *GTiff -- GeoTIFF File Format*
- GTIFF_READ_ANGULAR_PARAMS_IN_DEGREE: *GTiff -- GeoTIFF File Format*
- GTIFF_REPORT_COMPD_CS: *GTiff -- GeoTIFF File Format*
- GTIFF_VIRTUAL_MEM_IO: *GTiff -- GeoTIFF File Format*
- GTIFF_WRITE_ANGULAR_PARAMS_IN_DEGREE: *GTiff -- GeoTIFF File Format*
- GTIFF_WRITE_RAT_TO_PAM: *GTiff -- GeoTIFF File Format*
- GTIFF_WRITE_TOWGS84: *GTiff -- GeoTIFF File Format*
- GTI_NUM_THREADS: *GTI -- GDAL Raster Tile Index*
- HFA_COMPRESS_OVR: *HFA -- Erdas Imagine .img*
- HFA_USE_RRD: *HFA -- Erdas Imagine .img*
- IDB_OGR_FID: *IDB*
- INTERLEAVE_OVERVIEW:
  - *GTiff -- GeoTIFF File Format*
  - *gdaladdo*
- JP2KAK_FUSSY: *JP2KAK -- JPEG 2000 (based on Kakadu SDK)*
- JP2KAK_RESILIENT: *JP2KAK -- JPEG 2000 (based on Kakadu SDK)*
- JP2KAK_THREADS: *JP2KAK -- JPEG 2000 (based on Kakadu SDK)*
- JPEG_MASK_BIT_ORDER: *JPEG -- JPEG JFIF File Format*

- JPEG_QUALITY_OVERVIEW: *GTiff -- GeoTIFF File Format*
- JPEG_READ_MASK: *JPEG -- JPEG JFIF File Format*
- LIBKML_ALTITUDEMODE_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_BEGIN_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_BOTTOMFOV_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_CAMERA_ALTITUDEMODE_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_CAMERA_ALTITUDE_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_CAMERA_LONGITUDE_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_DESCRIPTION_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_DRAWORDER_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_END_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_EXTERNAL_STYLE: *LIBKML Driver (.kml .kmz)*
- LIBKML_EXTRUDE_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_HEADING_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_ICON_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_ID_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_IMAGEPYRAMID_GRIDORIGIN: *LIBKML Driver (.kml .kmz)*
- LIBKML_IMAGEPYRAMID_MAXHEIGHT: *LIBKML Driver (.kml .kmz)*
- LIBKML_IMAGEPYRAMID_MAXWIDTH: *LIBKML Driver (.kml .kmz)*
- LIBKML_IMAGEPYRAMID_TILESIZE: *LIBKML Driver (.kml .kmz)*
- LIBKML_LEFTFOV_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_MODEL_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_NAME_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_NEARFOV_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_NETWORKLINK_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_NETWORKLINK_FLYTOVIEW_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_NETWORKLINK_HTTPQUERY_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_NETWORKLINK_REFRESHINTERVAL_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_NETWORKLINK_REFRESHMODE_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_NETWORKLINK_REFRESHVISIBILITY_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_NETWORKLINK_VIEWBOUNDSCALE_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_NETWORKLINK_VIEWREFRESHMODE_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_NETWORKLINK_VIEWREFRESHTIME_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_PHOTOOVERLAY_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_PHOTOOVERLAY_SHAPE_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_READ_GROUND_OVERLAY: *LIBKML Driver (.kml .kmz)*

- LIBKML_RESOLVE_STYLE: *LIBKML Driver (.kml .kmz)*
- LIBKML_RIGHTFOV_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_ROLL_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_SCALE_X_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_SCALE_Y_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_SCALE_Z_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_SNIPPET_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_STYLEMAP_KEY: *LIBKML Driver (.kml .kmz)*
- LIBKML_TESSELLATE_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_TILT_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_TIMESTAMP_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_TOPFOV_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_USE_DOC.KML: *LIBKML Driver (.kml .kmz)*
- LIBKML_USE_SIMPLEFIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_VISIBILITY_FIELD: *LIBKML Driver (.kml .kmz)*
- LIBKML_WRAPDATELINE: *LIBKML Driver (.kml .kmz)*
- MAX_Z_ERROR_OVERVIEW: *GTiff -- GeoTIFF File Format*
- MBTILES_BAND_COUNT: *MBTiles*
- MITAB_BOUNDS_FILE: *MapInfo TAB and MIF/MID*
- MITAB_SET_TOWGS84_ON_KNOWN_DATUM: *MapInfo TAB and MIF/MID*
- MSSQLSPATIAL_ALWAYS_OUTPUT_FID: *MSSQLSpatial - Microsoft SQL Server Spatial Database*
- MSSQLSPATIAL_BCP_SIZE: *MSSQLSpatial - Microsoft SQL Server Spatial Database*
- MSSQLSPATIAL_LIST_ALL_TABLES: *MSSQLSpatial - Microsoft SQL Server Spatial Database*
- MSSQLSPATIAL_OGR_FID: *MSSQLSpatial - Microsoft SQL Server Spatial Database*
- MSSQLSPATIAL_SHOW_FID_COLUMN: *MSSQLSpatial - Microsoft SQL Server Spatial Database*
- MSSQLSPATIAL_USE_BCP: *MSSQLSpatial - Microsoft SQL Server Spatial Database*
- MSSQLSPATIAL_USE_GEOMETRY_COLUMNS: *MSSQLSpatial - Microsoft SQL Server Spatial Database*
- MSSQLSPATIAL_USE_GEOMETRY_VALIDATION: *MSSQLSpatial - Microsoft SQL Server Spatial Database*
- NGW_BATCH_SIZE: *NGW -- NextGIS Web*
- NGW_CACHE_EXPIRES: *NGW -- NextGIS Web*
- NGW_CACHE_MAX_SIZE: *NGW -- NextGIS Web*
- NGW_CONNECTTIMEOUT:
  - *NGW -- NextGIS Web*
  - *NGW -- NextGIS Web*
- NGW_EXTENSIONS: *NGW -- NextGIS Web*
- NGW_JSON_DEPTH:

  - *NGW -- NextGIS Web*
  - *NGW -- NextGIS Web*
- `NGW_MAX_RETRY`:
  - *NGW -- NextGIS Web*
  - *NGW -- NextGIS Web*
- `NGW_NATIVE_DATA`: *NGW -- NextGIS Web*
- `NGW_PAGE_SIZE`: *NGW -- NextGIS Web*
- `NGW_RETRY_DELAY`:
  - *NGW -- NextGIS Web*
  - *NGW -- NextGIS Web*
- `NGW_TIMEOUT`:
  - *NGW -- NextGIS Web*
  - *NGW -- NextGIS Web*
- `NGW_USERPWD`:
  - *NGW -- NextGIS Web*
  - *NGW -- NextGIS Web*
- `OCI_FID`: *Oracle Spatial*
- `OCI_PREFETCH_ROWS`: *Oracle Spatial*
- `OGR_APPLY_GEOM_SET_PRECISION`:
  - *GPKG -- GeoPackage vector*
  - *ogr2ogr*
  - *Configuration options*
- `OGR_ARC_MAX_GAP`:
  - *AutoCAD DWG*
  - *AutoCAD DXF*
  - *<literal>gdal vector set-geom-type</literal>*
  - *Configuration options*
- `OGR_ARC_STEPSIZE`:
  - *AutoCAD DWG*
  - *AutoCAD DXF*
  - *"INTERLIS 1" and "INTERLIS 2" drivers*
  - *<literal>gdal vector set-geom-type</literal>*
  - *Configuration options*
- `OGR_CT_USE_SRS_COORDINATE_EPOCH`:
  - *Configuration options*
  - *Coordinate epoch support*

- OGR_CURRENT_DATE:
  - *GPKG -- GeoPackage raster*
  - *GPKG -- GeoPackage vector*
- OGR_EDIGEO_CREATE_LABEL_LAYERS: *EDIGEO*
- OGR_EDIGEO_FONT_SIZE_FACTOR: *EDIGEO*
- OGR_ENABLE_PARTIAL_REPROJECTION: *Configuration options*
- OGR_FORCE_ASCII: *Configuration options*
- OGR_GEOJSON_ARRAY_AS_STRING: *GeoJSON*
- OGR_GEOJSON_DATE_AS_STRING: *GeoJSON*
- OGR_GEOJSON_MAX_OBJ_SIZE:
  - *GeoJSON*
  - *GeoJSONSeq: sequence of GeoJSON features*
- OGR_GPKG_NUM_THREADS: *GPKG -- GeoPackage vector*
- OGR_INTERLEAVED_READING: *OSM - OpenStreetMap XML and PBF*
- OGR_JSONFG_MAX_OBJ_SIZE: *JSONFG -- OGC Features and Geometries JSON*
- OGR_ODS_FIELD_TYPES: *ODS - Open Document Spreadsheet*
- OGR_ODS_HEADERS: *ODS - Open Document Spreadsheet*
- OGR_ORGANIZE_POLYGONS:
  - *ESRI Shapefile / DBF*
  - *Configuration options*
- OGR_PDF_READ_NON_STRUCTURED: *PDF -- Geospatial PDF*
- OGR_PG_CURSOR_PAGE: *PostgreSQL / PostGIS*
- OGR_PG_ENABLE_METADATA: *PostgreSQL / PostGIS*
- OGR_PG_RETRIEVE_FID:
  - *PostgreSQL / PostGIS*
  - *PostgreSQL / PostGIS - Advanced Driver Information*
- OGR_PG_SKIP_CONFLICTS:
  - *NAS - ALKIS*
  - *PostgreSQL / PostGIS*
- OGR_PROMOTE_TO_INTEGER64:
  - *SQLite / Spatialite RDBMS*
- OGR_SKIP: *Configuration options*
- OGR_SQLITE_CACHE:
  - *GPKG -- GeoPackage vector*
  - *SQLite / Spatialite RDBMS*
- OGR_SQLITE_JOURNAL:

  - *GPKG -- GeoPackage vector*
  - *SQLite / Spatialite RDBMS*
- `OGR_SQLITE_LIST_VIRTUAL_OGR`: *SQLite / Spatialite RDBMS*
- `OGR_SQLITE_LOAD_EXTENSIONS`:
  - *GPKG -- GeoPackage vector*
  - *SQLite / Spatialite RDBMS*
- `OGR_SQLITE_PRAGMA`:
  - *GPKG -- GeoPackage vector*
  - *SQLite / Spatialite RDBMS*
- `OGR_SQLITE_SYNCHRONOUS`:
  - *GPKG -- GeoPackage vector*
  - *SQLite / Spatialite RDBMS*
- `OGR_SQL_LIKE_AS_ILIKE`:
  - *Configuration options*
  - *OGR SQL dialect*
- `OGR_STROKE_CURVE`:
  - *"INTERLIS 1" and "INTERLIS 2" drivers*
  - *Configuration options*
- `OGR_STYLE`: *LIBKML Driver (.kml .kmz)*
- `OGR_TRUNCATE`: *PostgreSQL / PostGIS*
- `OGR_VFK_DB_DELETE`: *VFK - Czech Cadastral Exchange Data Format*
- `OGR_VFK_DB_NAME`: *VFK - Czech Cadastral Exchange Data Format*
- `OGR_VFK_DB_OVERWRITE`: *VFK - Czech Cadastral Exchange Data Format*
- `OGR_VFK_DB_READ`: *VFK - Czech Cadastral Exchange Data Format*
- `OGR_VFK_DB_READ_ALL_BLOCKS`: *VFK - Czech Cadastral Exchange Data Format*
- `OGR_VFK_DB_SPATIAL`: *VFK - Czech Cadastral Exchange Data Format*
- `OGR_WFS_BASE_START_INDEX`: *WFS - OGC WFS service*
- `OGR_WFS_PAGE_SIZE`: *WFS - OGC WFS service*
- `OGR_WFS_PAGING_ALLOWED`: *WFS - OGC WFS service*
- `OGR_WFS_USE_STREAMING`: *WFS - OGC WFS service*
- `OGR_WKT_PRECISION`: *Comma Separated Value (.csv)*
- `OGR_WKT_ROUND`: *Comma Separated Value (.csv)*
- `OGR_XLSX_FIELD_TYPES`: *XLSX - MS Office Open XML spreadsheet*
- `OGR_XLSX_HEADERS`: *XLSX - MS Office Open XML spreadsheet*
- `OGR_XLS_FIELD_TYPES`: *XLS - MS Excel format*
- `OGR_XLS_HEADERS`: *XLS - MS Excel format*

- OPENFILEGDB_DEFAULT_STRING_WIDTH: *ESRI File Geodatabase vector (OpenFileGDB)*
- OPENFILEGDB_IN_MEMORY_SPI: *ESRI File Geodatabase vector (OpenFileGDB)*
- OSM_COMPRESS_NODES: *OSM - OpenStreetMap XML and PBF*
- OSM_CONFIG_FILE: *OSM - OpenStreetMap XML and PBF*
- OSM_MAX_TMPFILE_SIZE: *OSM - OpenStreetMap XML and PBF*
- OSM_USE_CUSTOM_INDEXING: *OSM - OpenStreetMap XML and PBF*
- OSR_ADD_TOWGS84_ON_EXPORT_TO_PROJ4: *Configuration options*
- OSR_ADD_TOWGS84_ON_EXPORT_TO_WKT1: *Configuration options*
- OSR_ADD_TOWGS84_ON_IMPORT_FROM_EPSG: *Configuration options*
- OSR_DEFAULT_AXIS_MAPPING_STRATEGY: *Configuration options*
- OSR_STRIP_TOWGS84: *Configuration options*
- OSR_USE_NON_DEPRECATED: *Configuration options*
- OSR_WKT_FORMAT: *Configuration options*
- OSS_ACCESS_KEY_ID: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- OSS_ENDPOINT: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- OSS_SECRET_ACCESS_KEY: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- PDF_USER_PWD: *PDF -- Geospatial PDF*
- PGEO_DRIVER_TEMPLATE: *ESRI Personal GeoDatabase*
- PGSQL_OGR_FID: *PostgreSQL / PostGIS*
- PG_LIST_ALL_TABLES:
  - *PostgreSQL / PostGIS*
  - *PostgreSQL / PostGIS - Advanced Driver Information*
- PG_USE_BASE64: *PostgreSQL / PostGIS*
- PG_USE_COPY:
  - *PostgreSQL / PostGIS*
  - *PostgreSQL SQL Dump*
  - *ogr2ogr*
- PHOTOMETRIC_OVERVIEW:
  - *GTiff -- GeoTIFF File Format*
  - *gdaladdo*
- PLSCENES_PAGE_SIZE: *PLScenes (Planet Labs Scenes), Data V1 API*
- PL_API_KEY:
  - *PLMosaic (Planet Labs Mosaics API)*
  - *PLScenes (Planet Labs Scenes), Data V1 API*

- PREDICTOR_OVERVIEW: *GTiff -- GeoTIFF File Format*
- PYTHONSO:
  - *VRT -- GDAL Virtual Format*
  - *Configuration options*
- REPORT_COMPD_CS: *DTED -- Military Elevation Data*
- RPFTOC_FORCE_RGBA: *RPFTOC -- Raster Product Format/RPF (a.toc)*
- SHAPE_2GB_LIMIT: *ESRI Shapefile / DBF*
- SHAPE_ENCODING:
  - *ESRI Shapefile / DBF*
  - *Common options for vector programs*
- SHAPE_RESTORE_SHX: *ESRI Shapefile / DBF*
- SHAPE_REWIND_ON_WRITE: *ESRI Shapefile / DBF*
- SPARSE_OK_OVERVIEW: *GTiff -- GeoTIFF File Format*
- SQLITE_LIST_ALL_TABLES: *SQLite / Spatialite RDBMS*
- SQLITE_USE_OGR_VFS:
  - *GPKG -- GeoPackage vector*
  - *SQLite / Spatialite RDBMS*
- SSL_CERT_FILE:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- SWIFT_AUTH_TOKEN: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- SWIFT_AUTH_V1_URL: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- SWIFT_KEY: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- SWIFT_STORAGE_URL: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- SWIFT_USER: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- TAB_APPROX_GEOTRANSFORM: *GTiff -- GeoTIFF File Format*
- THRESHOLD: *Configuration options*
- TIFF_READ_STREAMING: *GTiff -- GeoTIFF File Format*
- TIFF_USE_OVR: *GTiff -- GeoTIFF File Format*
- USE_RRD:
  - *GTiff -- GeoTIFF File Format*
  - *gdaladdo*
  - *Configuration options*

- `USE_SPILL`: *HFA -- Erdas Imagine .img*
- `USE_TEMPFILE`: *GPSBabel*
- `USE_TILE_AS_BLOCK`: *JP2KAK -- JPEG 2000 (based on Kakadu SDK)*
- `VRT_NUM_THREADS`: *VRT -- GDAL Virtual Format*
- `VRT_VIRTUAL_OVERVIEWS`: *VRT -- GDAL Virtual Format*
- `VSIOSS_CHUNK_SIZE`: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `VSIS3_CHUNK_SIZE`: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `VSI_CACHE`:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `VSI_CACHE_SIZE`:
  - *Configuration options*
  - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `WASP_ADJ_TOLER`: *WAsP - WAsP .map format*
- `WASP_FIELDS`: *WAsP - WAsP .map format*
- `WASP_GEOM_FIELD`: *WAsP - WAsP .map format*
- `WASP_MERGE`: *WAsP - WAsP .map format*
- `WASP_POINT_TO_CIRCLE_RADIUS`: *WAsP - WAsP .map format*
- `WASP_TOLERANCE`: *WAsP - WAsP .map format*
- `WEBHDFS_DATANODE_HOST`: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `WEBHDFS_DELEGATION`: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `WEBHDFS_PERMISSION`: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `WEBHDFS_REPLICATION`: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `WEBHDFS_USERNAME`: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- `WEBP_LEVEL_OVERVIEW`: *GTiff -- GeoTIFF File Format*
- `WEBP_LOSSLESS_OVERVIEW`: *GTiff -- GeoTIFF File Format*
- `ZLEVEL_OVERVIEW`: *GTiff -- GeoTIFF File Format*
- `ZSTD_LEVEL_OVERVIEW`: *GTiff -- GeoTIFF File Format*

8.16 GDAL Python Utilities

The GDAL python utilities are included with GDAL. If you've installed GDAL you already have them. However you may want to use a newer or older version of the utilities without changing GDAL. This is where **gdal-utils** comes in.

gdal-utils: is the GDAL Python Utilities *distribution*. This is what you install. Its home page is https://pypi.org/project/gdal-utils/ . Install with `pip install gdal-utils`.

osgeo_utils: is the python *package*. This is what you use in your code after installing, e.g. `from osgeo_utils import ...`. If you're not writing code, ignore it.

Commonly used utilities include:

- gdal_merge
- gdal_edit
- gdal_calc
- ogrmerge

The sub-package `samples` contains the *Sample scripts*.

8.16.1 Developers

Read the gdal-utils project charter.

Clone or download the gdal project: https://github.com/OSGeo/gdal/

In your IDE set gdal-utils as the root folder, .../swig/python/gdal-utils.

> **./osgeo_utils** - contains the Programs (those scripts that have launch wrappers created by pip and added to PYTHONHOME/Scripts)
>
> **./osgeo_utils/samples** - working python scripts but not typically available in path (run them with `python3 path/to/samples/something.py`)

Improve the docs by editing the RST pages in .../doc/source which generate the web pages:

- https://gdal.org/api/index.html#python-api
- https://gdal.org/programs/index.html#programs
- https://gdal.org/api/python_samples.html

Contribute changes with Pull Requests from your fork to main GDAL project and use *gdal-utils* label.

8.17 Migration guide

8.17.1 From GDAL 3.13 to GDAL 3.14

- Changes impacting C++ users:
  - All methods accepting or returning `OGRBoolean` (aliased to `int`) have been changed to use `bool` instead.

8.17.2 From GDAL 3.12 to GDAL 3.13

- C API changes:
  - the following functions now return a `OGRErr` whereas they returned void before: *OGR_G_SetPointCount()*, *OGR_G_SetPoint()*, *OGR_G_SetPoint_2D()*, *OGR_G_SetPointM()*, *OGR_G_SetPointZM()*, *OGR_G_AddPoint()*, *OGR_G_AddPoint_2D()*, *OGR_G_AddPointM()*, *OGR_G_AddPointZM()*, *OGR_G_SetPoints()* and *OGR_G_SetPointsZM()*.

  - MIN, MAX and ABS pre-processor macros defined in cpl_port.h have been renamed to CPL_MIN, CPL_MAX and CPL_ABS.
  - M_PI is no longer exported. Users that would rely on this macro must now include math.h, with the _USE_MATH_DEFINES macro defined before.
- Changes impacting out-of-tree vector drivers:
  - *GDALDataset::Close()* takes now 2 input parameters (GDALProgressFunc pfnProgress, void *pProgressData), which may be nullptr.
  - *GDALDataset::AddBand()*, *GDALDataset::AdviseRead()*, *GDALDataset::BeginAsyncReader()*, *GDALDataset::CopyLayer()*, GDALDriver::pfnCreate, GDALDriver::pfnCreateCopy, *GDALRasterBand::AdviseRead()* and *GDALRasterBand::GetVirtualMemAuto()* now take a CSLConstList papszOptions parameter instead of char **.
- Changes impacting C++ users:
  - *GDALMajorObject::SetMetadata()* now takes a CSLConstList argument (this does not require code changes but is an opportunity for users to have better const safety)
  - *GDALMajorObject::GetMetadata()* and *GDALGetMetadata()* now return a CSLConstList argument. This will require users that stored the return value of those functions in char ** to use CSLConstList instead. Such change is compatible with earlier GDAL versions.
- GDAL CLI changes:
  - Several command-line arguments in the unified GDAL CLI were renamed from a --src/--dst pattern to an --input/--output pattern. The old argument names are still accepted by command line tools, C, C++ and Python API.
- Behavior changes:
  - RasterIO resampling/VRT: do it by default in the output buffer type unless new field GDALRasterIOExtraArg::bOperateInBufType in set to false (#14221). For example, now by default is using non-nearest resampling in RasterIO() to read a Byte band into a Float32 buffer, the result will generally be non-integer values, matching what is assumed to be the default wished behavior.

8.17.3 From GDAL 3.11 to GDAL 3.12

- The following changes have been done to the "gdal" command line interface:
  - Sub-commands "buffer", "explode-collections", "make-valid", "segmentize", "simplify", "swap-xy" of "gdal vector geom" are now directly available under "gdal vector". Support for the old location is kept for 3.12, but will be definitely removed in 3.13.
  - Furthermore, sub-command "set-type" of "gdal vector geom" is renamed as "set-geom-type" and also placed under "gdal vector".
  - Progress bar is emitted to stdout by default unless --quiet/-q is specified.
  - For gdal raster info, gdal vector info and gdal vsi list, --format=text is now the default when those utilities are invoked from the command line. When they are invoked from the API, the default is still JSON.
- Changes impacting out-of-tree vector drivers: * *GDALDataset::GetLayer()* is now a const method that returns a const OGRLayer* * *GDALDataset::GetLayerCount()* is now a const method * *GDALDataset::TestCapability()* is now a const method * *OGRLayer::GetName()* is now a const method * *OGRLayer::GetGeomType()* is now a const method * *OGRLayer::GetLayerDefn()* is now a const method that returns a const OGRFeatureDefn* * *OGRLayer::GetFIDColumn()* is now a const method * *OGRLayer::GetGeometryColumn()* is now a const method * *OGRLayer::GetSpatialRef()* is now a const method that returns a const OGRSpatialReference* * *OGRLayer::TestCapability()* is now a const method

- Changes impacting C++ users:
  - *OGRLayer::GetSpatialRef()* now returns a `const OGRSpatialReference*` instead of a non-const one. In most situations, users can modify their code to store the returned value in a const pointer (which should be compatible with older GDAL versions). When they need to modify the reference count of the returned instance, const casting to a non-const pointer is the current recommended solution.
  - *OGRFeature::GetDefnRef()* now returns a `const OGRFeatureDefn*` instead of a non-const one. Same recommendation as above.
- C API change: The *GDALRATFieldType* enumeration has been extended with 3 new values: *GDALRATFieldType::GFT_Boolean*, *GDALRATFieldType::GFT_DateTime*, *GFT_WKBGeometry*. Code that calls *GDALRATGetTypeOfCol()* may encounter those new values.
- The raw file capabilities (`VRTRawRasterBand`) of the VRT raster driver have been limited by default for security reasons. Consult *Restricting access to Raw Files* for more details.
- Methods *GDALRasterAttributeTable::SetValue()* now return a CPLErr instead of void. This will impact in particular out-of-tree drivers that implement those methods in a subclass of *GDALRasterAttributeTable*.
- Virtual methods *GDALDataset::GetGeoTransform()* (resp. *GDALDataset::SetGeoTransform()* have been modified to accept `GDALGeoTransform& gt` (resp. `const GDALGeoTransform& gt`) parameters instead of a pointer to 6 doubles. The new class *GDALGeoTransform* is a thin wrapper around a `std::array<double, 6>`. This change affects out-of-tree raster drivers.

8.17.4 From GDAL 3.10 to GDAL 3.11

- The introduction of the gdal_fwd.h header that normalizes forward declarations of GDAL public opaque types may cause issues with downstream users of the GDAL API that would have redefined themselves those types, particularly when building against a GDAL built in DEBUG mode where the type aliases are stricter than in release mode.
- The `OGRLayer::GetExtent(OGREnvelope*, int bForce)` and `OGRLayer::GetExtent(int iGeomField, OGREnvelope*, int bForce)` methods are no longer virtual methods that are implemented by drivers, and the `int bForce` parameter is now a `bool bForce`. Drivers may implement the new `OGRLayer::IGetExtent(int, OGREnvelope *, bool)` protected virtual method. The public method checks that the iGeomField value is in range. Similarly for `OGRLayer::GetExtent3D(int iGeomField, OGREnvelope3D*, int bForce)` which is now a user facing method (with the change that the `int bForce` is now a `bool bForce`). Drivers may implement the new `IGetExtent3D(int iGeomField, OGREnvelope3D*, bool bForce)` protected virtual method. The public method checks that the iGeomField value is in range.
- The *OGRLayer::SetSpatialFilter()* and *OGRLayer::SetSpatialFilterRect()* methods are no longer virtual methods that are implemented by drivers. They now return OGRErr instead of void, and accept a `const OGRGeometry*`). Drivers may implement the new ISetSpatialFilter(int iGeomField, const OGRGeometry*) protected virtual method. The public methods check that the iGeomField value is in range.
- GDAL drivers may now return raster bands with the new data types GDT_Float16 or GDT_CFloat16. Code that use the GDAL API must be ready to react to the new data type, possibly by doing RasterIO() requests with `eBufType==GDT_Float32`, if they can't deal natively with Float16 values.
- If only a specific GDAL Minor version is to be supported, this must now be specified in the find_package call in CMake via a version range specification.
- The following methods `OGRCoordinateTransformation::Transform(size_t nCount, double *x, double *y, double *z, double *t, int *pabSuccess)` and `OGRCoordinateTransformation::TransformWithErrorCodes(size_t nCount, double *x, double *y, double *z, double *t, int *panErrorCodes)` are modified to return FALSE as soon as at least one point fails to transform (to be consistent with the other form of `Transform()` that doesn't take a "t" argument), whereas previously they would return FALSE only if no transformation was found. When

FALSE is returned the `pabSuccess[]` or `panErrorCodes[]` arrays indicate which point succeeded or failed to transform.

The `GDALTransformerFunc` callback and its implementations (`GenImgProjTransformer`, `RPCTransformer`, etc.) are also modified to return FALSE as soon as at least one point fails to transform.

8.17.5 From GDAL 3.9 to GDAL 3.10

- The OGR SQL parser has been modified to evaluate NULL values in boolean operations similarly to other SQL engines (SQLite, PostgreSQL, etc.). Previously, with a foo=NULL field, expressions `foo NOT IN ('bar')` and `foo NOT LIKE ('bar')` would evaluate as true. Now the result is false (with the NULL state being propagated to it). Concretely, to get the same results as in previous versions, the above expressions must be rewritten as `foo IS NULL OR foo NOT IN ('bar')` and `foo IS NULL OR foo NOT LIKE ('bar')`.
- MEM driver: opening a dataset with the `MEM:::` syntax is now disabled by default because of security implications. This can be enabled by setting the GDAL_MEM_ENABLE_OPEN build or configuration option. Creation of a 0-band MEM dataset, and using the *`GDALDataset::AddBand()`* method with the DATAPOINTER, PIXELOFFSET and LINEOFFSET options is the recommended way. For example, like in https://github.com/OSGeo/gdal/blob/e32a2fde41a555b7948cece9ab9b4e979138e7dd/gcore/rasterio.cpp#L1534-L1576
- The Erdas Imagine (HFA) and Derived drivers are now optional drivers. Users building with GDAL_BUILD_OPTIONAL_DRIVERS=OFF may need to explicitly enable them with GDAL_ENABLE_DRIVER_HFA=ON and GDAL_ENABLE_DRIVER_DERIVED=ON. The MapInfo, OGR_VRT and KML drivers are now an optional driver. Users building with OGR_BUILD_OPTIONAL_DRIVERS=OFF may need to explicitly enable them with OGR_ENABLE_DRIVER_TAB=ON, OGR_ENABLE_DRIVER_VRT=ON and OGR_ENABLE_DRIVER_KML=ON.
- User code using *`VSIFEofL()`* to potentially to end read loops should also test the return code of the new *`VSIFErrorL()`* function. Some virtual file systems that used to report errors through `VSIFEofL` now do through `VSIFErrorL`.
- Out-of-tree implementations of *`VSIVirtualHandle`*: 2 new required virtual methods must be implemented: `int Error()`, and `void ClearErr()` following POSIX semantics of `ferror()` and `clearerr()`. This is to distinguish `Read()` that returns less bytes than requested because of an error (`Error() != 0`) or because of end-of-file (`Eof() != 0`)

  The `VSIFilesystemPluginCallbacksStruct` structure is extended with 2 corresponding optional (but recommended to be implemented to reliably detect reading errors) callbacks "error" and "clear_err".
- Python bindings: *`osgeo.gdal.Band.GetStatistics()`* and *`osgeo.gdal.Band.ComputeStatistics()`* now return a None value in case of error (when exceptions are not enabled)
- New color interpretation (`GCI_xxxx`) items have been added to the *`GDALColorInterp`* enumeration. Code testing color interpretation may need to be adapted.

8.17.6 From GDAL 3.8 to GDAL 3.9

- Out-of-tree vector drivers:
  - *`OGRLayer::CreateField()`* now takes a `const OGRFieldDefn*` instead of a `OGRFieldDefn*`.
  - *`OGRLayer::CreateGeomField()`* now takes a `const OGRGeomFieldDefn*` instead of a `OGRGeomFieldDefn*`.
  - `GDALDataset::ICreateLayer()` has a new prototype, due to RFC 99 "Geometry coordinate precision changes.

    The fastest migration path is from:

```
OGRLayer *
     MyDataset::ICreateLayer(const char* pszLayerName,
                             const OGRSpatialReference *poSpatialRef,
                             OGRwkbGeometryType eGType, char **papszOptions)
{
    ...
}
```

to

```
OGRLayer *
     MyDataset::ICreateLayer(const char *pszLayerName,
                             const OGRGeomFieldDefn *poGeomFieldDefn,
                             CSLConstList papszOptions)
{
    const auto eGType = poGeomFieldDefn ? poGeomFieldDefn->GetType() : wkbNone;
    const auto poSpatialRef =
        poGeomFieldDefn ? poGeomFieldDefn->GetSpatialRef() : nullptr;
    ...
}
```

- Sealed feature and field definition (RFC 97). A number of drivers now "seal" their layer definition, which might cause issue to user code currently mis-using setters of OGRFeatureDefn, OGRFieldDefn or OGRGeomFieldDefn on such instances. The drivers that have been updated to seal their layer definition are: GeoPackage, PostgreSQL, Shapefile, OpenFileGDB, MITAB, Memory, GeoJSON, JSONFG, TopoJSON, ESRIJSON, ODS, XLSX.
- OGRLayer::SetIgnoredFields() now accepts a `CSLConstList papszIgnoredFields` instead of a `const char** papszIgnoredFields`

8.17.7 From GDAL 3.7 to GDAL 3.8

- Out-of-tree vector drivers:
  - `GDALDataset::ICreateLayer()` now takes a `const OGRSpatialReference*` instead of a `OGRSpatialReference*`. Drivers should clone the passed SRS if they need to keep it.
- The /vsimem virtual file system is modified to automatically create parent directories when a file is created. (e.g., creating /vsimem/parent/child.txt would cause the directory /vsimem/parent to be created.) If the parent directory cannot be created because the file /vsimem/parent exists, file creation will now fail.
- In SWIG bindings, the function FileFromMemBuffer now returns an error code if the file could not be created.

8.17.8 From GDAL 3.6 to GDAL 3.7

- Following RFC 87, PIXELTYPE=SIGNEDBYTE in IMAGE_STRUCTURE metadata domain is no longer reported by drivers that used to do it. The new GDT_Int8 data type is now reported. On writing, the PIXELTYPE=SIGNEDBYTE creation option is preserved in drivers that used to support it, but is deprecated and external code should rather use the GDT_Int8 data type.
- The VSILFILE* type is no longer aliased to FILE* in builds without the DEBUG define (that is production builds). External code that used FILE* with GDAL VSI*L API has to be changed to use VSILFILE*. This aliasing dates back to old times where both types were indifferently used in the code base. In the mean time, this was cleaned up. But there was a drawback of having VSILFILE* being either a dedicated type or an alias of FILE* depending whether DEBUG is defined, especially with the C++ API, for people building their plugins with DEBUG and running them against a non-DEBUG GDAL build, or the reverse.

- GDALFlushCache() and GDALDataset::FlushCache() are modified to return a CPLErr error code instead of void. Affects out-of-tree drivers.
- A Close() virtual method is added to GDALDataset per RFC 91. Out-of-tree drivers with write support are encouraged to implement it for error propagation.
- Pansharpening now requires that panchromatic and multispectral bands have valid geotransform (in early versions, it was assumed in the case of missing geotransform that they covered the same geospatial extent). The undocumented VRT pansharpened MSShiftX and MSShiftY options (and the corresponding C++ `GDALPansharpenOptions::dfMSShiftX` and `dfMSShiftY` members) have been removed, due to using the inverted convention as one would expect, and being sub-par solution compared to using geotransform to correlate pixels of panchromatic and multispectral bands.
- OGRCoordinateTransformation::GetSourceCS() and GetTargetCS() now returns a const OGRSpatialReference*
- OGRGeometry::getSpatialReference() now returns a const OGRSpatialReference*
- OGRGeomFieldDefn::GetSpatialRef() now returns a const OGRSpatialReference*

8.17.9 From GDAL 3.5 to GDAL 3.6

- Out-of-tree vector drivers:
  - GDALDataset::IBuildOverviews(): parameters `panOverviewList` and `panBandList` are now of type 'const int*' (previously 'int*') Added a CSLConstList papszOptions member.
  - GDALRasterBand::BuildOverviews(): parameter `panOverviewList` is now of type 'const int*' (previously 'int*') Added a CSLConstList papszOptions member.
  - Compatibility layers of GDAL 3.0 `_GetProjectionRef()`, `_GetGCPProjectionRef()`, `_SetProjection()`, `_SetGCPs()` have been removed

8.17.10 From GDAL 3.4 to GDAL 3.5

- GDAL drivers may now return raster bands with the new data types GDT_Int64 or GDT_UInt64.
- Make GDALProxyRasterBand::RefUnderlyingRasterBand() / UnrefUnderlyingRasterBand() const. May affect out-of-tree drivers

8.17.11 From GDAL 3.3 to GDAL 3.4

- Out-of-tree vector drivers:
  - *`OGRFeatureDefn`* protected member variables have been changed.
    - `(nFieldCount, papoFieldDefn) ==> std::vector<std::unique_ptr<OGRFieldDefn>> apoFieldDefn{}`
    - `(nGeomFieldCount, paoGeomFieldDefn) ==> std::vector<std::unique_ptr<OGRGeomFieldDefn>> apoGeomFieldDefn{}`
  - `OGRFeatureDefn::AddGeomFieldDefn( OGRGeomFieldDefn *, bCopy = FALSE )` is replaced by `AddGeomFieldDefn( std::unique_ptr<OGRGeomFieldDefn>&& )`
  - *`GDALDataset::FlushCache()`* and *`GDALRasterBand::FlushCache()`* now takes a `bool bAtClosing` argument. That argument is set to true when FlushCache() is called from the dataset/band destructor. This can be used as a hint, for example to avoid doing extra work if the dataset is marked for deletion at closing. Driver implementing that method should propagate the argument to the base implementation when calling it.

8.17.12 From GDAL 3.2 to GDAL 3.3

- Python bindings:
  - Python 2 is no longer supported per RFC 77. Python 3.6 or later required
  - "osgeo.utils" was replaced by "osgeo_utils" (more details: see RFC78)
  - The following undocumented, untested utility scripts are no longer installed as system scripts and were moved from "gdal/swig/python/gdal-utils" to: "gdal/swig/python/gdal-utils/samples":
    * `epsg_tr.py`
    * `esri2wkt.py`
    * `gcps2vec.py`
    * `gcps2wld.py`
    * `gdal_auth.py`
    * `gdalchksum.py`
    * `gdalident.py`
    * `gdalimport.py`
    * `mkgraticule.py`

    In order to import sample script X (i.e. *epsg_tr*) as a module, use: *from osgeo_utils.samples import X*. In order to run it as a script run: *python -m osgeo_utils.samples.X*.
  - packaging:
    * "gdal/swig/python/samples" moved to: "gdal/swig/python/gdal-utils/osgeo_utils/samples"
    * "gdal/swig/python/scripts" moved to: "gdal/swig/python/gdal-utils/scripts"
- gdaldem TRI: default algorithm has been changed to use Riley et al. 1999. Use -alg Wilson to select the algorithm used previously
- Disable by default raster drivers DODS, JPEG2000(Jasper), JPEGLS, MG4LIDAR, FUJIBAS, IDA, INGR, ZMAP and vector driver ARCGEN, ArcObjects, CLOUDANT, COUCHDB, DB2, DODS, FME, GEOMEDIA, GTM, INGRES, MONGODB, REC, WALK at runtime, unless the GDAL_ENABLE_DEPRECATED_DRIVER_{drivername} configuration option is set to YES. Those drivers are planned for complete removal in GDAL 3.5
- Perl bindings are deprecated. Removal planned for GDAL 3.5. Use `Geo::GDAL::FFI` instead
- Removal of BNA, AeronavFAA, HTF, OpenAir, SEGUKOOA, SEGY, SUA, XPlane, BPG, E00GRID, EPSILON, IGNFHeightASCIIGrid, NTV1 drivers. Moved to (unsupported) https://github.com/OSGeo/gdal-extra-drivers repository.

8.17.13 From GDAL 3.1 to GDAL 3.2

- Python bindings: old-style, deprecated for many years, import method of importing the gdal module through "import gdal" is no longer available. "from osgeo import gdal" must now be used. This holds true for the `ogr`, `osr`, `gdalconst` and `gdalnumeric` modules.

8.17.14 From GDAL 3.0 to GDAL 3.1

- OGR SQL: the 'LIKE' operator is now case sensitive by default. ILIKE (supported in previous versions) must be used for case insensitive comparison. The OGR_SQL_LIKE_AS_ILIKE configuration option can be set to YES to make LIKE behave in a case insensitive way as in previous versions.

8.17.15 From GDAL 2.x to GDAL 3.0

Consult https://github.com/OSGeo/gdal/blob/v3.0.4/gdal/MIGRATION_GUIDE.TXT

8.18 Security considerations

This page discusses some security issues that users or developers may face while using GDAL. It can also serve as a check-list of items to take care of, for developers coding new drivers.

While GDAL maintainers take a number of steps to limit the risk of software vulnerabilities, GDAL has a large attack surface, because it deals with data in a variety of file formats.

> **Note**
>
> This page is not intended to be a comprehensive cook-book for being "safe" in all circumstances, and should be considered as work-in-progress that can benefit from continued feedback of people deploying GDAL. Although they are specialists of the GDAL code base, GDAL maintainers are not security experts.

8.18.1 Classes of potential vulnerabilities

- Arbitrary code execution with the privileges of the user running the GDAL process.
- Theft, tampering or destruction of data.
- Denial of Service : abortion of a process, high consumption of CPU, memory, I/O resource.
- Unwanted network access.

8.18.2 Causes

- Software bugs - in GDAL code itself, or in third-party dependencies - related to code that processes user data. Common defects can be:
  - stack or heap buffer overflows, potentially leading to arbitrary code execution.
  - excessive memory allocation
  - infinite, or very long, loops in code
- Software functionalities themselves, depending on the context of use, can be a source of vulnerabilities. Consult *Credential related issues* below.

8.18.3 Situations at risk

The following are examples of situations where a user may process untrusted data that has been crafted to trigger a vulnerability:

- web services accepting input files provided by a client,
- desktop use of GDAL where an hostile party manages to convince the victim user to process hostile data.

8.18.4 Mitigation

- Upgrade to the latest versions of GDAL and its dependencies that might contain bug fixes for some vulnerabilities.
- Build GDAL with the hardening options of compilers, e.g. with -D_FORTIFY_SOURCE=2 (some Linux distributions turn on it by default), to minimize the effect of buffer overflows.

- Restrict access to the part of the file system that are only needed. The `chroot` mechanism, or other sand-boxing solutions, might be a technical solution for this.
- Compile only the subset of drivers needed. The `GDAL_ENABLE_DRIVER_{name}` or `OGR_ENABLE_DRIVER_{name}` CMake build options can be used for that purpose.
- Disable unneeded drivers at run-time by setting the *GDAL_SKIP* configuration option.
- When network access is not needed, disable Curl at build time with the `GDAL_USE_CURL=OFF` CMake option.
- Process untrusted data in a dedicated user account with no access to other local sensitive data (data files, passwords, encryption/signing keys, etc...). Note that the mere act of opening a dataset with the GDAL/OGR API or utilities is a form of processing.
- For automatic services, place restrictions on the CPU time and memory consumption allowed to the process using GDAL. Check the options of your HTTP server.
- Check the raster dimensions and number of bands just after opening a GDAL driver and before processing it. Several GDAL drivers use the `GDALCheckDatasetDimensions()` and `GDALCheckBandCount()` functions to do early sanity checks, but user checks can also be useful. For example, a dataset with huge raster dimensions but with a very small file size might be suspicious (but not always, for example a VRT file, or highly compressed data...)
- Do not allow arbitrary arguments to be passed to command line utilities. In particular `--config GDAL_DRIVER_PATH xxx` or `--config OGR_DRIVER_PATH xxx` could be used to trigger arbitrary code execution for someone with write access to some parts of the file system. Similarly, *GDAL_VRT_ENABLE_PYTHON* or *GDAL_VRT_PYTHON_TRUSTED_MODULES* could be used to trigger hostile Python code execution.
- When possible, avoid using closed-source dependency libraries that cannot be audited for vulnerabilities.

8.18.5 Credential related issues

GDAL network virtual file systems can access credentials needed to grant access to remote resources, from the environment or local files. Those secrets are stored in RAM using normal mechanisms. Consequently they could potentially be recovered by untrusted code that would run in the same process as GDAL code.

The same applies for connection strings to some drivers such as PostgreSQL, MySQL, etc. which may include passwords.

8.18.6 Known issues in API

- *`OGRSpatialReference::SetFromUserInput()`* accepts URLs by default

8.18.7 Known issues in drivers

8.18.7.1 General issues

- Drivers do not always use file extensions to determine which file must be handled by which driver (this is a feature in most situations). But, for example, a VRT file might be disguised as a .tif, .png, or .jpg file. So you cannot know which driver will handle a file by just looking at its extension. Using `gdalmanage identify the.file` (or *`GDALIdentifyDriver()`*) can be a way to know the driver without attempting a full open of the file, but, drivers not having a specialized implementation of the Identify() method will fall back to the Open() method.
- Drivers depending on third-party libraries whose code has been embedded in GDAL. Binary builds might rely on the internal version, or the external version. If using the internal version, they might use an obsolete version of the third-party library that might contain known vulnerabilities. Potentially concerned drivers are GTiff (libtiff, libgeotiff), PNG (libpng), GIF (giflib), JPEG (libjpeg), PCRaster (libcsf), GeoJSON (libjson-c), MRF (liblerc). An internal version of ZLib is also contained in GDAL sources. Packagers of GDAL are recommended to use the external version of the libraries when possible, so that security upgrades of those dependencies benefit to GDAL.

- Drivers using GDALOpen()/GDALOpenEx()/OGROpen() internally cause other drivers to be used (and their possible flaws exploited), without it being obvious at first sight. VRT, MBTiles, KMLSuperOverlay, RasterLite, PCIDSK, PDF, RPFTOC, RS2, WMS, WCS, WFS, OAPIF, OGCAPI and GTI are examples of drivers with this behavior.
- Drivers depending on downloaded data (HTTP, WMS, WCS, WFS, OAPIF, OGCAPI, STACIT, STACTA, etc.).
- XML based drivers: they might be subject to denial of service by billion laugh-like attacks. Existing GDAL XML based drivers are thought to take defensive measures against such patterns (starting with GDAL 3.9.3 for LVBAG and GMLAS drivers)
- SQL injections: services that would accept untrusted SQL requests could trigger SQL injection vulnerabilities in database-based drivers.

8.18.7.2 GDAL MEM driver

The opening syntax `MEM:::DATAPOINTER=<memory_address>,PIXELS=<number>,LINES=<number>` can access any memory of the process. Feeding it with a random access can cause a crash, or a read of unwanted virtual memory. The MEM driver is used by various algorithms and drivers in creation mode (which is not vulnerable to the DATAPOINTER issue), so completely disabling the driver might be detrimental to other areas of GDAL. It is possible to define the GDAL_NO_OPEN_FOR_MEM_DRIVER *compilation* flag to disable the `MEM:::DATAPOINTER` syntax only.

8.18.7.3 GDAL PDF driver

The OGR_DATASOURCE creation option accepts a file name. So any OGR datasource, and potentially any file (see OGR VRT) could be read through this option, and its content embedded in the generated PDF. Similarly for the COMPOSITION_FILE creation option.

8.18.7.4 GDAL VRT driver

It can be used to access any valid GDAL dataset. If a hostile party, with knowledge of the location on the filesystem of a valid GDAL dataset, convinces a user to run gdal_translate a VRT file and give it back the result, it might be able to steal data. That could potentially be able for a web service accepting data from the user, converting it, and sending back the result.

The VRTRawRasterBand mechanism can read any file (not necessarily a valid GDAL dataset) accessible, which can extend the scope of the above mentioned issue. Consult *Restricting access to Raw Files* for more details.

The VRTDerivedRasterBand mechanism can use Pixel functions written in Python, directly embedded in a VRT file, or pointing to external Python code. By default this mechanism is disabled, to avoid arbitrary code execution. Consult *Security implications* for more details.

/vsicurl/ (and associated network-capable virtual file systems) filenames can be used, thus causing remote data to be downloaded.

8.18.7.5 GDAL GTI driver

Same issues as the GDAL VRT driver.

8.18.7.6 OGR VRT driver

Similar issues as the GDAL VRT driver. `<SrcSQL>` could be used to modify data.

CHAPTER

NINE

API

API is omitted in this PDF document. You can consult it on https://gdal.org/api/index.html

CHAPTER

TEN

TUTORIALS

10.1 Raster

10.1.1 Raster API tutorial

10.1.1.1 Opening the File

Before opening a GDAL supported raster datastore it is necessary to register drivers. There is a driver for each supported format. Normally this is accomplished with the *GDALAllRegister()* function which attempts to register all known drivers, including those auto-loaded from .so files using *GDALDriverManager::AutoLoadDrivers()*. If for some applications it is necessary to limit the set of drivers it may be helpful to review the code from gdalallregister.cpp. Python automatically calls GDALAllRegister() when the gdal module is imported.

Once the drivers are registered, the application should call the free standing *GDALOpen()* function to open a dataset, passing the name of the dataset and the access desired (GA_ReadOnly or GA_Update).

C++

C

Python

```
// Note: since GDAL >= 3.12, "gdal_raster_cpp.h" can also be used
// or finer grained include files such as "gdal_drivermanager.h",
// "gdal_dataset.h" and "gdal_rasterband.h".
#include "gdal_priv.h"

#include <errno.h>

int main(int argc, const char* argv[])
{
    if (argc != 2) {
        return EINVAL;
    }
    const char* pszFilename = argv[1];

    GDALDatasetUniquePtr poDataset;
    GDALAllRegister();
    const GDALAccess eAccess = GA_ReadOnly;
    poDataset = GDALDatasetUniquePtr(GDALDataset::FromHandle(GDALOpen( pszFilename,␣
↪eAccess )));
    if( !poDataset )
    {
        ...; // handle error
```

(continues on next page)

(continued from previous page)

```
    }
    return 0;
}
```

```
#include "gdal.h"
#include "cpl_conv.h" /* for CPLMalloc() */

#include <errno.h>

int main(int argc, const char* argv[])
{
    if (argc != 2) {
        return EINVAL;
    }
    const char* pszFilename = argv[1];

    GDALDatasetH  hDataset;
    GDALAllRegister();
    const GDALAccess eAccess = GA_ReadOnly;
    hDataset = GDALOpen( pszFilename, eAccess );
    if( hDataset == NULL )
    {
        ...; // handle error
    }
    return 0;
}
```

```
from osgeo import gdal
dataset = gdal.Open(filename, gdal.GA_ReadOnly)
if not dataset:
    ...
```

Note that if *GDALOpen()* returns NULL it means the open failed, and that an error messages will already have been emitted via *CPLError()*. If you want to control how errors are reported to the user review the *CPLError()* documentation. Generally speaking all of GDAL uses *CPLError()* for error reporting. Also, note that pszFilename need not actually be the name of a physical file (though it usually is). It's interpretation is driver dependent, and it might be an URL, a filename with additional parameters added at the end controlling the open or almost anything. Please try not to limit GDAL file selection dialogs to only selecting physical files.

10.1.1.2 Getting Dataset Information

As described in the *Raster Data Model*, a `GDALDataset` contains a list of raster bands, all pertaining to the same area, and having the same resolution. It also has metadata, a coordinate system, a georeferencing transform, size of raster and various other information.

In the particular, but common, case of a "north up" image without any rotation or shearing, the georeferencing transform *Geotransform Tutorial* takes the following form :

```
adfGeoTransform[0] /* top left x */
adfGeoTransform[1] /* w-e pixel resolution */
adfGeoTransform[2] /* 0 */
adfGeoTransform[3] /* top left y */
```

(continues on next page)

(continued from previous page)

```
adfGeoTransform[4] /* 0 */
adfGeoTransform[5] /* n-s pixel resolution (negative value) */
```

In the general case, this is an affine transform.

If we wanted to print some general information about the dataset we might do the following:

C++

C

Python

```
double        adfGeoTransform[6];
printf( "Driver: %s/%s\n",
        poDataset->GetDriver()->GetDescription(),
        poDataset->GetDriver()->GetMetadataItem( GDAL_DMD_LONGNAME ) );
printf( "Size is %dx%dx%d\n",
        poDataset->GetRasterXSize(), poDataset->GetRasterYSize(),
        poDataset->GetRasterCount() );
if( poDataset->GetProjectionRef()  != NULL )
    printf( "Projection is `%s'\n", poDataset->GetProjectionRef() );
if( poDataset->GetGeoTransform( adfGeoTransform ) == CE_None )
{
    printf( "Origin = (%.6f,%.6f)\n",
            adfGeoTransform[0], adfGeoTransform[3] );
    printf( "Pixel Size = (%.6f,%.6f)\n",
            adfGeoTransform[1], adfGeoTransform[5] );
}
```

```
GDALDriverH   hDriver;
double        adfGeoTransform[6];
hDriver = GDALGetDatasetDriver( hDataset );
printf( "Driver: %s/%s\n",
        GDALGetDriverShortName( hDriver ),
        GDALGetDriverLongName( hDriver ) );
printf( "Size is %dx%dx%d\n",
        GDALGetRasterXSize( hDataset ),
        GDALGetRasterYSize( hDataset ),
        GDALGetRasterCount( hDataset ) );
if( GDALGetProjectionRef( hDataset ) != NULL )
    printf( "Projection is `%s'\n", GDALGetProjectionRef( hDataset ) );
if( GDALGetGeoTransform( hDataset, adfGeoTransform ) == CE_None )
{
    printf( "Origin = (%.6f,%.6f)\n",
            adfGeoTransform[0], adfGeoTransform[3] );
    printf( "Pixel Size = (%.6f,%.6f)\n",
            adfGeoTransform[1], adfGeoTransform[5] );
}
```

```
print("Driver: {}/{}".format(dataset.GetDriver().ShortName,
                             dataset.GetDriver().LongName))
print("Size is {} x {} x {}".format(dataset.RasterXSize,
                                    dataset.RasterYSize,
```

(continues on next page)

(continued from previous page)

```
                                    dataset.RasterCount))
print("Projection is {}".format(dataset.GetProjection()))
geotransform = dataset.GetGeoTransform()
if geotransform:
    print("Origin = ({}, {})".format(geotransform[0], geotransform[3]))
    print("Pixel Size = ({}, {})".format(geotransform[1], geotransform[5]))
```

10.1.1.3 Fetching a Raster Band

At this time access to raster data via GDAL is done one band at a time. Also, there is metadata, block sizes, color tables, and various other information available on a band by band basis. The following codes fetches a *GDALRasterBand* object from the dataset (numbered 1 through *GDALDataset::GetRasterCount()*) and displays a little information about it.

C++

C

Python

```
GDALRasterBand  *poBand;
int             nBlockXSize, nBlockYSize;
int             bGotMin, bGotMax;
double          adfMinMax[2];
poBand = poDataset->GetRasterBand( 1 );
poBand->GetBlockSize( &nBlockXSize, &nBlockYSize );
printf( "Block=%dx%d Type=%s, ColorInterp=%s\n",
        nBlockXSize, nBlockYSize,
        GDALGetDataTypeName(poBand->GetRasterDataType()),
        GDALGetColorInterpretationName(
            poBand->GetColorInterpretation()) );
adfMinMax[0] = poBand->GetMinimum( &bGotMin );
adfMinMax[1] = poBand->GetMaximum( &bGotMax );
if( ! (bGotMin && bGotMax) )
    GDALComputeRasterMinMax((GDALRasterBandH)poBand, TRUE, adfMinMax);
printf( "Min=%.3fd, Max=%.3f\n", adfMinMax[0], adfMinMax[1] );
if( poBand->GetOverviewCount() > 0 )
    printf( "Band has %d overviews.\n", poBand->GetOverviewCount() );
if( poBand->GetColorTable() != NULL )
    printf( "Band has a color table with %d entries.\n",
            poBand->GetColorTable()->GetColorEntryCount() );
```

```
GDALRasterBandH hBand;
int             nBlockXSize, nBlockYSize;
int             bGotMin, bGotMax;
double          adfMinMax[2];
hBand = GDALGetRasterBand( hDataset, 1 );
GDALGetBlockSize( hBand, &nBlockXSize, &nBlockYSize );
printf( "Block=%dx%d Type=%s, ColorInterp=%s\n",
        nBlockXSize, nBlockYSize,
        GDALGetDataTypeName(GDALGetRasterDataType(hBand)),
        GDALGetColorInterpretationName(
            GDALGetRasterColorInterpretation(hBand)) );
```

(continues on next page)

(continued from previous page)

```
adfMinMax[0] = GDALGetRasterMinimum( hBand, &bGotMin );
adfMinMax[1] = GDALGetRasterMaximum( hBand, &bGotMax );
if( ! (bGotMin && bGotMax) )
    GDALComputeRasterMinMax( hBand, TRUE, adfMinMax );
printf( "Min=%.3fd, Max=%.3f\n", adfMinMax[0], adfMinMax[1] );
if( GDALGetOverviewCount(hBand) > 0 )
    printf( "Band has %d overviews.\n", GDALGetOverviewCount(hBand));
if( GDALGetRasterColorTable( hBand ) != NULL )
    printf( "Band has a color table with %d entries.\n",
            GDALGetColorEntryCount(
                GDALGetRasterColorTable( hBand ) ) );
```

```
band = dataset.GetRasterBand(1)
print("Band Type={}".format(gdal.GetDataTypeName(band.DataType)))

min = band.GetMinimum()
max = band.GetMaximum()
if not min or not max:
    (min,max) = band.ComputeRasterMinMax(True)
print("Min={:.3f}, Max={:.3f}".format(min,max))

if band.GetOverviewCount() > 0:
    print("Band has {} overviews".format(band.GetOverviewCount()))

if band.GetRasterColorTable():
    print("Band has a color table with {} entries".format(band.GetRasterColorTable().
↪GetCount()))
```

10.1.1.4 Reading Raster Data

There are a few ways to read raster data, but the most common is via the *GDALRasterBand::RasterIO()* method. This method will automatically take care of data type conversion, up/down sampling and windowing. The following code will read the first scanline of data into a similarly sized buffer, converting it to floating point as part of the operation.

The pafScanline buffer should be freed with CPLFree() when it is no longer used.

C++

C

Python

```
float *pafScanline;
int   nXSize = poBand->GetXSize();
pafScanline = (float *) CPLMalloc(sizeof(float)*nXSize);
poBand->RasterIO( GF_Read, 0, 0, nXSize, 1,
                pafScanline, nXSize, 1, GDT_Float32,
                0, 0 );
```

```
float *pafScanline;
int   nXSize = GDALGetRasterBandXSize( hBand );
pafScanline = (float *) CPLMalloc(sizeof(float)*nXSize);
GDALRasterIO( hBand, GF_Read, 0, 0, nXSize, 1,
```

(continues on next page)

(continued from previous page)

```
            pafScanline, nXSize, 1, GDT_Float32,
            0, 0 );
```

```
scanline = band.ReadRaster(xoff=0, yoff=0,
                        xsize=band.XSize, ysize=1,
                        buf_xsize=band.XSize, buf_ysize=1,
                        buf_type=gdal.GDT_Float32)
```

Note that when using Python the returned scanline is of type string, and contains xsize*4 bytes of raw binary floating point data. This can be converted to Python values using the struct module from the standard library:

```
import struct
tuple_of_floats = struct.unpack('f' * b2.XSize, scanline)
```

The RasterIO call takes the following arguments.

```
CPLErr GDALRasterBand::RasterIO( GDALRWFlag eRWFlag,
                                 int nXOff, int nYOff, int nXSize, int nYSize,
                                 void * pData, int nBufXSize, int nBufYSize,
                                 GDALDataType eBufType,
                                 int nPixelSpace,
                                 int nLineSpace )
```

Note that the same RasterIO() call is used to read, or write based on the setting of eRWFlag (either GF_Read or GF_Write). The nXOff, nYOff, nXSize, nYSize argument describe the window of raster data on disk to read (or write). It doesn't have to fall on tile boundaries though access may be more efficient if it does.

The pData is the memory buffer the data is read into, or written from. It's real type must be whatever is passed as eBufType, such as GDT_Float32, or GDT_UInt8. The RasterIO() call will take care of converting between the buffer's data type and the data type of the band. Note that when converting floating point data to integer RasterIO() rounds down, and when converting source values outside the legal range of the output the nearest legal value is used. This implies, for instance, that 16bit data read into a GDT_UInt8 buffer will map all values greater than 255 to 255, the data is not scaled!

The nBufXSize and nBufYSize values describe the size of the buffer. When loading data at full resolution this would be the same as the window size. However, to load a reduced resolution overview this could be set to smaller than the window on disk. In this case the RasterIO() will utilize overviews to do the IO more efficiently if the overviews are suitable.

The nPixelSpace, and nLineSpace are normally zero indicating that default values should be used. However, they can be used to control access to the memory data buffer, allowing reading into a buffer containing other pixel interleaved data for instance.

10.1.1.5 Closing the Dataset

Please keep in mind that *GDALRasterBand* objects are owned by their dataset, and they should never be destroyed with the C++ delete operator. *GDALDataset*'s can be closed by calling *GDALClose()* (it is NOT recommended to use the delete operator on a GDALDataset for Windows users because of known issues when allocating and freeing memory across module boundaries. See the relevant topic on the FAQ). Calling GDALClose will result in proper cleanup, and flushing of any pending writes. Forgetting to call GDALClose on a dataset opened in update mode in a popular format like GTiff will likely result in being unable to open it afterwards.

10.1.1.6 Techniques for Creating Files

New files in GDAL supported formats may be created if the format driver supports creation. There are two general techniques for creating files, using CreateCopy() and Create(). The CreateCopy method involves calling the CreateCopy() method on the format driver, and passing in a source dataset that should be copied. The Create method involves calling the Create() method on the driver, and then explicitly writing all the metadata, and raster data with separate calls. All drivers that support creating new files support the CreateCopy() method, but only a few support the Create() method.

To determine if a particular format supports Create or CreateCopy it is possible to check the DCAP_CREATE and DCAP_CREATECOPY metadata on the format driver object. Ensure that *GDALAllRegister()* has been called before calling *GDALDriverManager::GetDriverByName()*. In this example we fetch a driver, and determine whether it supports Create() and/or CreateCopy().

C++

C

Python

```
#include "cpl_string.h"
...
const char *pszFormat = "GTiff";
GDALDriver *poDriver;
char **papszMetadata;
poDriver = GetGDALDriverManager()->GetDriverByName(pszFormat);
if( poDriver == NULL )
    exit( 1 );
papszMetadata = poDriver->GetMetadata();
if( CSLFetchBoolean( papszMetadata, GDAL_DCAP_CREATE, FALSE ) )
    printf( "Driver %s supports Create() method.\n", pszFormat );
if( CSLFetchBoolean( papszMetadata, GDAL_DCAP_CREATECOPY, FALSE ) )
    printf( "Driver %s supports CreateCopy() method.\n", pszFormat );
```

```
#include "cpl_string.h"
...
const char *pszFormat = "GTiff";
GDALDriverH hDriver = GDALGetDriverByName( pszFormat );
char **papszMetadata;
if( hDriver == NULL )
    exit( 1 );
papszMetadata = GDALGetMetadata( hDriver, NULL );
if( CSLFetchBoolean( papszMetadata, GDAL_DCAP_CREATE, FALSE ) )
    printf( "Driver %s supports Create() method.\n", pszFormat );
if( CSLFetchBoolean( papszMetadata, GDAL_DCAP_CREATECOPY, FALSE ) )
    printf( "Driver %s supports CreateCopy() method.\n", pszFormat );
```

```
fileformat = "GTiff"
driver = gdal.GetDriverByName(fileformat)
metadata = driver.GetMetadata()
if metadata.get(gdal.DCAP_CREATE) == "YES":
    print("Driver {} supports Create() method.".format(fileformat))

if metadata.get(gdal.DCAP_CREATECOPY) == "YES":
    print("Driver {} supports CreateCopy() method.".format(fileformat))
```

Note that a number of drivers are read-only and won't support Create() or CreateCopy().

10.1.1.7 Using CreateCopy()

The *GDALDriver::CreateCopy()* method can be used fairly simply as most information is collected from the source dataset. However, it includes options for passing format specific creation options, and for reporting progress to the user as a long dataset copy takes place. A simple copy from the a file named pszSrcFilename, to a new file named pszDstFilename using default options on a format whose driver was previously fetched might look like this:

C++

C

Python

```
GDALDataset *poSrcDS =
(GDALDataset *) GDALOpen( pszSrcFilename, GA_ReadOnly );
GDALDataset *poDstDS;
poDstDS = poDriver->CreateCopy( pszDstFilename, poSrcDS, FALSE,
                                NULL, NULL, NULL );
/* Once we're done, close properly the dataset */
if( poDstDS != NULL )
    GDALClose( (GDALDatasetH) poDstDS );
GDALClose( (GDALDatasetH) poSrcDS );
```

```
GDALDatasetH hSrcDS = GDALOpen( pszSrcFilename, GA_ReadOnly );
GDALDatasetH hDstDS;
hDstDS = GDALCreateCopy( hDriver, pszDstFilename, hSrcDS, FALSE,
                         NULL, NULL, NULL );
/* Once we're done, close properly the dataset */
if( hDstDS != NULL )
    GDALClose( hDstDS );
GDALClose(hSrcDS);
```

```
src_ds = gdal.Open(src_filename)
dst_ds = driver.CreateCopy(dst_filename, src_ds, strict=0)
# Once we're done, close properly the dataset
dst_ds = None
src_ds = None
```

Note that the CreateCopy() method returns a writable dataset, and that it must be closed properly to complete writing and flushing the dataset to disk. In the Python case this occurs automatically when "dst_ds" goes out of scope. The FALSE (or 0) value used for the bStrict option just after the destination filename in the CreateCopy() call indicates that the CreateCopy() call should proceed without a fatal error even if the destination dataset cannot be created to exactly match the input dataset. This might be because the output format does not support the pixel datatype of the input dataset, or because the destination cannot support writing georeferencing for instance.

A more complex case might involve passing creation options, and using a predefined progress monitor like this:

C++

C

Python

```
#include "cpl_string.h"
...
```

(continues on next page)

(continued from previous page)

```
char **papszOptions = NULL;
papszOptions = CSLSetNameValue( papszOptions, "TILED", "YES" );
papszOptions = CSLSetNameValue( papszOptions, "COMPRESS", "PACKBITS" );
poDstDS = poDriver->CreateCopy( pszDstFilename, poSrcDS, FALSE,
                                papszOptions, GDALTermProgress, NULL );
/* Once we're done, close properly the dataset */
if( poDstDS != NULL )
    GDALClose( (GDALDatasetH) poDstDS );
CSLDestroy( papszOptions );
```

```
#include "cpl_string.h"
...
char **papszOptions = NULL;
papszOptions = CSLSetNameValue( papszOptions, "TILED", "YES" );
papszOptions = CSLSetNameValue( papszOptions, "COMPRESS", "PACKBITS" );
hDstDS = GDALCreateCopy( hDriver, pszDstFilename, hSrcDS, FALSE,
                         papszOptions, GDALTermProgres, NULL );
/* Once we're done, close properly the dataset */
if( hDstDS != NULL )
    GDALClose( hDstDS );
CSLDestroy( papszOptions );
```

```
src_ds = gdal.Open(src_filename)
dst_ds = driver.CreateCopy(dst_filename, src_ds, strict=0,
                         options=["TILED=YES", "COMPRESS=PACKBITS"])
# Once we're done, close properly the dataset
dst_ds = None
src_ds = None
```

10.1.1.8 Using Create()

For situations in which you are not just exporting an existing file to a new file, it is generally necessary to use the *GDALDriver::Create()* method (though some interesting options are possible through use of virtual files or in-memory files). The Create() method takes an options list much like CreateCopy(), but the image size, number of bands and band type must be provided explicitly.

C++

C

Python

```
GDALDataset *poDstDS;
char **papszOptions = NULL;
poDstDS = poDriver->Create( pszDstFilename, 512, 512, 1, GDT_UInt8,
                            papszOptions );
```

```
GDALDatasetH hDstDS;
char **papszOptions = NULL;
hDstDS = GDALCreate( hDriver, pszDstFilename, 512, 512, 1, GDT_UInt8,
                     papszOptions );
```

```
dst_ds = driver.Create(dst_filename, xsize=512, ysize=512,
                       bands=1, eType=gdal.GDT_UInt8)
```

Once the dataset is successfully created, all appropriate metadata and raster data must be written to the file. What this is will vary according to usage, but a simple case with a projection, geotransform and raster data is covered here.

C++

C

Python

```
double adfGeoTransform[6] = { 444720, 30, 0, 3751320, 0, -30 };
OGRSpatialReference oSRS;
char *pszSRS_WKT = NULL;
GDALRasterBand *poBand;
GByte abyRaster[512*512];
poDstDS->SetGeoTransform( adfGeoTransform );
oSRS.SetUTM( 11, TRUE );
oSRS.SetWellKnownGeogCS( "NAD27" );
oSRS.exportToWkt( &pszSRS_WKT );
poDstDS->SetProjection( pszSRS_WKT );
CPLFree( pszSRS_WKT );
poBand = poDstDS->GetRasterBand(1);
poBand->RasterIO( GF_Write, 0, 0, 512, 512,
                 abyRaster, 512, 512, GDT_UInt8, 0, 0 );
/* Once we're done, close properly the dataset */
GDALClose( (GDALDatasetH) poDstDS );
```

```
double adfGeoTransform[6] = { 444720, 30, 0, 3751320, 0, -30 };
OGRSpatialReferenceH hSRS;
char *pszSRS_WKT = NULL;
GDALRasterBandH hBand;
GByte abyRaster[512*512];
GDALSetGeoTransform( hDstDS, adfGeoTransform );
hSRS = OSRNewSpatialReference( NULL );
OSRSetUTM( hSRS, 11, TRUE );
OSRSetWellKnownGeogCS( hSRS, "NAD27" );
OSRExportToWkt( hSRS, &pszSRS_WKT );
OSRDestroySpatialReference( hSRS );
GDALSetProjection( hDstDS, pszSRS_WKT );
CPLFree( pszSRS_WKT );
hBand = GDALGetRasterBand( hDstDS, 1 );
GDALRasterIO( hBand, GF_Write, 0, 0, 512, 512,
            abyRaster, 512, 512, GDT_UInt8, 0, 0 );
/* Once we're done, close properly the dataset */
GDALClose( hDstDS );
```

```
from osgeo import osr
import numpy
dst_ds.SetGeoTransform([444720, 30, 0, 3751320, 0, -30])
srs = osr.SpatialReference()
srs.SetUTM(11, 1)
srs.SetWellKnownGeogCS("NAD27")
```

(continues on next page)

(continued from previous page)

```
dst_ds.SetProjection(srs.ExportToWkt())
raster = numpy.zeros((512, 512), dtype=numpy.uint8)
dst_ds.GetRasterBand(1).WriteArray(raster)
# Once we're done, close properly the dataset
dst_ds = None
```

10.1.2 Raster driver implementation tutorial

10.1.2.1 Overall Approach

In general new formats are added to GDAL by implementing format specific drivers as subclasses of *GDALDataset*, and band accessors as subclasses of *GDALRasterBand*. As well, a *GDALDriver* instance is created for the format, and registered with the *GDALDriverManager*, to ensure that the system knows about the format.

This tutorial will start with implementing a simple read-only driver (based on the JDEM driver), and then proceed to utilizing the RawRasterBand helper class, implementing creatable and updatable formats, and some esoteric issues.

It is strongly advised that the *Raster Data Model* be reviewed and understood before attempting to implement a GDAL driver.

10.1.2.2 Implementing the Dataset

We will start showing minimal implementation of a read-only driver for the Japanese DEM format (jdemdataset.cpp). First we declare a format specific dataset class, JDEMDataset in this case.

```
class JDEMDataset final: public GDALPamDataset
{
    friend class JDEMRasterBand;

    VSILFILE           *m_fp = nullptr;
    GByte               m_abyHeader[HEADER_SIZE];
    OGRSpatialReference m_oSRS{};

  public:
                       JDEMDataset();
                      ~JDEMDataset();

    static GDALDataset *Open( GDALOpenInfo * );
    static int Identify( GDALOpenInfo * );

    CPLErr GetGeoTransform( double * padfTransform ) override;
    const OGRSpatialReference* GetSpatialRef() const override;
};
```

In general we provide capabilities for a driver, by overriding the various virtual methods on the GDALDataset base class. However, the Open() method is special. This is not a virtual method on the base class, and we will need a freestanding function for this operation, so we declare it static. Implementing it as a method in the JDEMDataset class is convenient because we have privileged access to modify the contents of the database object.

The open method itself may look something like this:

```
GDALDataset *JDEMDataset::Open( GDALOpenInfo *poOpenInfo )
{
    // Confirm that the header is compatible with a JDEM dataset.
```

(continues on next page)

(continued from previous page)

```
    if (!Identify(poOpenInfo))
        return nullptr;

    // Confirm the requested access is supported.
    if( poOpenInfo->eAccess == GA_Update )
    {
        CPLError(CE_Failure, CPLE_NotSupported,
                 "The JDEM driver does not support update access to existing "
                 "datasets.");
        return nullptr;
    }

    // Check that the file pointer from GDALOpenInfo* is available.
    if( poOpenInfo->fpL == nullptr )
    {
        return nullptr;
    }

    // Create a corresponding GDALDataset.
    auto poDS = cpl::make_unique<JDEMDataset>();

    // Borrow the file pointer from GDALOpenInfo*.
    std::swap(poDS->m_fp, poOpenInfo->fpL);

    // Store the header (we have already checked it is at least HEADER_SIZE
    // byte large).
    memcpy(poDS->m_abyHeader, poOpenInfo->pabyHeader, HEADER_SIZE);

    const char *psHeader = reinterpret_cast<const char *>(poDS->m_abyHeader);
    poDS->nRasterXSize = JDEMGetField(psHeader + 23, 3);
    poDS->nRasterYSize = JDEMGetField(psHeader + 26, 3);
    if( !GDALCheckDatasetDimensions(poDS->nRasterXSize, poDS->nRasterYSize) )
    {
        return nullptr;
    }

    // Create band information objects.
    poDS->SetBand(1, new JDEMRasterBand(poDS.get(), 1));

    // Initialize any PAM information.
    poDS->SetDescription(poOpenInfo->pszFilename);
    poDS->TryLoadXML();

    // Check for overviews.
    poDS->oOvManager.Initialize(poDS.get(), poOpenInfo->pszFilename);

    return poDS.release();
}
```

The first step in any database Open function is to verify that the file being passed is in fact of the type this driver is for. It is important to realize that each driver's Open function is called in turn till one succeeds. Drivers must quietly return nullptr if the passed file is not of their format. They should only produce an error if the file does appear to be of their supported format, but is for some reason unsupported or corrupt. The information on the file to be opened is passed in

contained in a GDALOpenInfo object. The GDALOpenInfo includes the following public data members:

```
char        *pszFilename;
char**      papszOpenOptions;
GDALAccess  eAccess;  // GA_ReadOnly or GA_Update
int         nOpenFlags;
int         bStatOK;
int         bIsDirectory;
VSILFILE   *fpL;
int         nHeaderBytes;
GByte       *pabyHeader;
```

The driver can inspect these to establish if the file is supported. If the *pszFilename* refers to an object in the file system, the *bStatOK* flag will be set to TRUE. As well, if the file was successfully opened, the first kilobyte or so is read in, and put in pabyHeader, with the exact size in *nHeaderBytes*.

In this typical testing example it is verified that the file was successfully opened, that we have at least enough header information to perform our test, and that various parts of the header are as expected for this format. In this case, there are no magic numbers for JDEM format so we check various date fields to ensure they have reasonable century values. If the test fails, we quietly return NULL indicating this file isn't of our supported format.

The identification is in fact delegated to a Identify() static function :

```
/************************************************************************/
/*                              Identify()                              */
/************************************************************************/
int JDEMDataset::Identify( GDALOpenInfo * poOpenInfo )
{
    if( poOpenInfo->nHeaderBytes < HEADER_SIZE )
        return FALSE;

    // Confirm that the header has what appears to be dates in the
    // expected locations.
    // Check if century values seem reasonable.
    const char *psHeader = reinterpret_cast<char *>(poOpenInfo->pabyHeader);
    if( (!STARTS_WITH_CI(psHeader + 11, "19") &&
         !STARTS_WITH_CI(psHeader + 11, "20")) ||
        (!STARTS_WITH_CI(psHeader + 15, "19") &&
         !STARTS_WITH_CI(psHeader + 15, "20")) ||
        (!STARTS_WITH_CI(psHeader + 19, "19") &&
         !STARTS_WITH_CI(psHeader + 19, "20")) )
    {
        return FALSE;
    }

    // Check the extent too. In particular, that we are in the first quadrant,
    // as this is only for Japan.
    const double dfLLLat = JDEMGetAngle(psHeader + 29);
    const double dfLLLong = JDEMGetAngle(psHeader + 36);
    const double dfURLat = JDEMGetAngle(psHeader + 43);
    const double dfURLong = JDEMGetAngle(psHeader + 50);
    if( dfLLLat > 90 || dfLLLat < 0 ||
        dfLLLong > 180 ||dfLLLong < 0 ||
        dfURLat > 90 || dfURLat < 0 ||
        dfURLong > 180 || dfURLong < 0 ||
```

(continues on next page)

(continued from previous page)

```
        dfLLLat > dfURLat ||
        dfLLLong > dfURLong )
    {
        return FALSE;
    }

    return TRUE;
}
```

It is important to make the "is this my format" test as stringent as possible. In this particular case, we check that dates are in the 19th or 20th centry, but as this might also be too weak, we check that the geospatial extent is consistent, and valid for Japan. Once we are satisfied that the file is of our format, we can do any other tests that are necessary to validate the file is usable, and in particular that we can provide the level of access desired. Since the JDEM driver does not provide update support, error out in that case.

```
if( poOpenInfo->eAccess == GA_Update )
{
    CPLError(CE_Failure, CPLE_NotSupported,
             "The JDEM driver does not support update access to existing "
             "datasets.");
    return NULL;
}
```

Next we need to create an instance of the database class in which we will set various information of interest. We create it as a std::unique_ptr<JDEMDataset> with the cpl::make_unique<> utility (equivalent to std::make_unique<> available in C++14 and later), to make memory management easier in error code paths.

```
// Check that the file pointer from GDALOpenInfo* is available.
if( poOpenInfo->fpL == NULL )
{
    return NULL;
}
auto poDS = cpl::make_unique<JDEMDataset>();

// Borrow the file pointer from GDALOpenInfo*.
std::swap(poDS->m_fp, poOpenInfo->fpL);
```

At this point we "borrow" the file handle that was held by GDALOpenInfo* (we did make sure that poDS->m_fp is initialized to nullptr in the inline member definition). This file pointer uses the VSI*L GDAL API to access files on disk. This virtualized POSIX-style API allows some special capabilities like supporting large files, in-memory files and zipped files.

Next the X and Y size are extracted from the header. The *nRasterXSize* and *nRasterYSize* are data fields inherited from the GDALDataset base class, and must be set by the Open() method.

```
// Store the header (we have already checked it is at least HEADER_SIZE
// byte large).
memcpy(poDS->m_abyHeader, poOpenInfo->pabyHeader, HEADER_SIZE);

const char *psHeader = reinterpret_cast<const char *>(poDS->m_abyHeader);
poDS->nRasterXSize = JDEMGetField(psHeader + 23, 3);
poDS->nRasterYSize = JDEMGetField(psHeader + 26, 3);
if( !GDALCheckDatasetDimensions(poDS->nRasterXSize, poDS->nRasterYSize) )
```

(continues on next page)

(continued from previous page)

```
{
    return nullptr;
}
```

All the bands related to this dataset must be created and attached using the SetBand() method. We will explore the JDEMRasterBand() class shortly.

```
// Create band information objects.
poDS->SetBand(1, new JDEMRasterBand(poDS.get(), 1));
```

Finally we assign a name to the dataset object, and call the GDALPamDataset TryLoadXML() method which can initialize auxiliary information from an .aux.xml file if available. We also initialize for external overviews (in a .ovr side car file). For more details on these services review the GDALPamDataset and related classes.

```
    // Initialize any PAM information.
    poDS->SetDescription( poOpenInfo->pszFilename );
    poDS->TryLoadXML();

    // Check for overviews.
    poDS->oOvManager.Initialize(poDS.get(), poOpenInfo->pszFilename);

    return poDS.release();
}
```

10.1.2.3 Implementing the RasterBand

Similar to the customized JDEMDataset class subclassed from GDALDataset, we also need to declare and implement a customized JDEMRasterBand derived from *GDALRasterBand* for access to the band(s) of the JDEM file. For JDEMRasterBand the declaration looks like this:

```
class JDEMRasterBand final: public GDALPamRasterBand
{
    friend class JDEMDataset;

    int          m_nRecordSize = 0;
    char        *m_pszRecord = nullptr;
    bool         m_bBufferAllocFailed = false;

  public:
                JDEMRasterBand( JDEMDataset *, int );
               ~JDEMRasterBand();

    virtual CPLErr IReadBlock( int, int, void * ) override;
};
```

The constructor may have any signature, and is only called from the Open() method. Other virtual methods, such as *GDALRasterBand::IReadBlock()* must be exactly matched to the method signature in gdal_priv.h.

The constructor implementation looks like this:

```
JDEMRasterBand::JDEMRasterBand( JDEMDataset *poDSIn, int nBandIn ) :
    // Cannot overflow as nBlockXSize <= 999.
    m_nRecordSize(poDSIn->GetRasterXSize() * 5 + 9 + 2)
```

(continues on next page)

(continued from previous page)

```
{
    poDS = poDSIn;
    nBand = nBandIn;

    eDataType = GDT_Float32;

    nBlockXSize = poDS->GetRasterXSize();
    nBlockYSize = 1;
}
```

The following data members are inherited from GDALRasterBand, and should generally be set in the band constructor.

```
poDS: Pointer to the parent GDALDataset.
nBand: The band number within the dataset.
eDataType: The data type of pixels in this band.
nBlockXSize: The width of one block in this band.
nBlockYSize: The height of one block in this band.
```

The full set of possible GDALDataType values are declared in gdal.h, and include GDT_UInt8, GDT_UInt16, GDT_Int16, and GDT_Float32. The block size is used to establish a natural or efficient block size to access the data with. For tiled datasets this will be the size of a tile, while for most other datasets it will be one scanline, as in this case.

Next we see the implementation of the code that actually reads the image data, IReadBlock().

```
CPLErr JDEMRasterBand::IReadBlock( int /* nBlockXOff */,
                                   int nBlockYOff,
                                   void * pImage )

{
    JDEMDataset *poGDS = cpl::down_cast<JDEMDataset *>(poDS);

    if (m_pszRecord == nullptr)
    {
        if (m_bBufferAllocFailed)
            return CE_Failure;

        m_pszRecord = static_cast<char *>(VSI_MALLOC_VERBOSE(m_nRecordSize));
        if (m_pszRecord == nullptr)
        {
            m_bBufferAllocFailed = true;
            return CE_Failure;
        }
    }

    CPL_IGNORE_RET_VAL(
        VSIFSeekL(poGDS->m_fp, 1011 + m_nRecordSize * nBlockYOff, SEEK_SET));

    if( VSIFReadL(m_pszRecord, m_nRecordSize, 1, poGDS->m_fp) != 1 )
    {
        CPLError(CE_Failure, CPLE_AppDefined,
                 "Cannot read scanline %d", nBlockYOff);
        return CE_Failure;
    }
```

(continues on next page)

(continued from previous page)

```
    if( !EQUALN(reinterpret_cast<char *>(poGDS->m_abyHeader), m_pszRecord, 6) )
    {
        CPLError(CE_Failure, CPLE_AppDefined,
                 "JDEM Scanline corrupt.  Perhaps file was not transferred "
                 "in binary mode?");
        return CE_Failure;
    }

    if( JDEMGetField(m_pszRecord + 6, 3) != nBlockYOff + 1 )
    {
        CPLError(CE_Failure, CPLE_AppDefined,
                 "JDEM scanline out of order, JDEM driver does not "
                 "currently support partial datasets.");
        return CE_Failure;
    }

    for( int i = 0; i < nBlockXSize; i++ )
        static_cast<float *>(pImage)[i] =
            JDEMGetField(m_pszRecord + 9 + 5 * i, 5) * 0.1f;

    return CE_None;
}
```

Key items to note are:

- It is typical to cast the GDALRasterBand::poDS member to the derived type of the owning dataset. If your RasterBand class will need privileged access to the owning dataset object, ensure it is declared as a friend (omitted above for brevity).
- If an error occurs, report it with CPLError(), and return CE_Failure. Otherwise return CE_None.
- The pImage buffer should be filled with one block of data. The block is the size declared in nBlockXSize and nBlockYSize for the raster band. The type of the data within pImage should match the type declared in eDataType in the raster band object.
- The nBlockXOff and nBlockYOff are block offsets, so with 128x128 tiled datasets values of 1 and 1 would indicate the block going from (128,128) to (255,255) should be loaded.

10.1.2.4 The Driver

While the JDEMDataset and JDEMRasterBand are now ready to use to read image data, it still isn't clear how the GDAL system knows about the new driver. This is accomplished via the *GDALDriverManager*. To register our format we implement a registration function. The declaration goes in gcore/gdal_frmts.h: void CPL_DLL GDALRegister_JDEM(void);

The definition in the driver file is:

```
void GDALRegister_JDEM()

{
    if( !GDAL_CHECK_VERSION("JDEM") )
        return;

    if( GDALGetDriverByName("JDEM") != nullptr )
```

(continues on next page)

(continued from previous page)

```
        return;

    GDALDriver *poDriver = new GDALDriver();

    poDriver->SetDescription("JDEM");
    poDriver->SetMetadataItem(GDAL_DCAP_RASTER, "YES");
    poDriver->SetMetadataItem(GDAL_DMD_LONGNAME, "Japanese DEM (.mem)");
    poDriver->SetMetadataItem(GDAL_DMD_HELPTOPIC, "drivers/raster/jdem.html");
    poDriver->SetMetadataItem(GDAL_DMD_EXTENSIONS, "mem");
    poDriver->SetMetadataItem(GDAL_DCAP_VIRTUALIO, "YES");

    poDriver->pfnOpen = JDEMDataset::Open;
    poDriver->pfnIdentify = JDEMDataset::Identify;

    GetGDALDriverManager()->RegisterDriver(poDriver);
}
```

Note the use of GDAL_CHECK_VERSION macro. This is a macro that should be used by drivers that can be built as a plugin. As the GDAL C++ ABI may, and will, change between GDAL feature releases (for example from GDAL 3.x.0 to 3.y.0), it is necessary to recompile your driver against the header files of the GDAL feature version with which you want to make it work. The GDAL_CHECK_VERSION macro will check that the GDAL version with which the driver was compiled and the version against which it is running are compatible (checking that the major and minor version numbers are equal). The C++ ABI will however remain stable for releases of the same release branch (that is for bug fixes releases x.y.z of a given feature release x.y.0).

The registration function will create an instance of a GDALDriver object when first called, and register it with the GDALDriverManager. The following fields can be set in the driver before registering it with the GDALDriverManager.

- The description is the short name for the format. This is a unique name for this format, often used to identity the driver in scripts and command line programs. Normally 3-5 characters in length, and matching the prefix of the format classes. (mandatory)
- GDAL_DCAP_RASTER: set to YES to indicate that this driver handles raster data. (mandatory)
- GDAL_DMD_LONGNAME: A longer descriptive name for the file format, but still no longer than 50-60 characters. (mandatory)
- GDAL_DMD_HELPTOPIC: The name of a help topic to display for this driver, if any. In this case JDEM format is contained within the various format web page held in gdal/html. (optional)
- GDAL_DMD_EXTENSIONS: The extensions used for files of this type, without the leading '.'. If more than one, they should be separated with space. (optional)
- GDAL_DMD_MIMETYPE: The standard mime type for this file format, such as "image/png". (optional)
- GDAL_DMD_CREATIONOPTIONLIST: There is evolving work on mechanisms to describe creation options. See the geotiff driver for an example of this. (optional)
- GDAL_DMD_CREATIONDATATYPES: A list of space separated data types supported by this create when creating new datasets. If a Create() method exists, these will be will supported. If a CreateCopy() method exists, this will be a list of types that can be losslessly exported but it may include weaker data types than the type eventually written. For instance, a format with a CreateCopy() method, and that always writes Float32 might also list Byte, Int16, and UInt16 since they can losslessly translated to Float32. An example value might be "Byte Int16 UInt16". (required - if creation supported)
- GDAL_DCAP_VIRTUALIO: set to YES to indicate that this driver can deal with files opened with the VSI*L GDAL API. Otherwise this metadata item should not be defined. (optional)

- pfnOpen: The function to call to try opening files of this format. (optional)
- pfnIdentify: The function to call to try identifying files of this format. A driver should return 1 if it recognizes the file as being of its format, 0 if it recognizes the file as being NOT of its format, or -1 if it cannot reach to a firm conclusion by just examining the header bytes. (optional)
- pfnCreate: The function to call to create new updatable datasets of this format. (optional)
- pfnCreateCopy: The function to call to create a new dataset of this format copied from another source, but not necessary updatable. (optional)
- pfnDelete: The function to call to delete a dataset of this format. (optional)
- pfnUnloadDriver: A function called only when the driver is destroyed. Could be used to cleanup data at the driver level. Rarely used. (optional)

For a driver that can be built as a plugin (that is to say a standalone shared object, loaded at runtime by GDAL), since GDAL 3.9 and rfc-96, there is a way to implement the driver in a way where the plugin will be loaded only when necessary, and not immediately at *GDALAllRegister()* time. Consult rfc96_example_driver for the changes needed to make the driver compatible with deferred plugin loading.

10.1.2.5 Adding Driver to GDAL Tree

Note that the GDALRegister_JDEM() method must be called by the higher level program in order to have access to the JDEM driver. Normal practice when writing new drivers is to:

- Add a driver directory under frmts, with the directory name the same as the short name.
- Add a CMakeLists.txt in that directory modeled on those from other similar directories (i.e. the jdem directory).
- Reference the new driver in frmts/CMakeLists.txt, using the gdal_optional_format() or gdal_dependent_format() functions depending if it requires no external dependency or it has at least one.
- Add the module with the dataset, and rasterband implementation. Generally this is called <short_name>dataset.cpp, with all the GDAL specific code in one file, though that is not required.
- Add the registration entry point declaration (i.e. GDALRegister_JDEM()) to gcore/gdal_frmts.h.
- Add a call to the registration function to frmts/gdalallregister.cpp, protected by an appropriate #ifdef.

Once this is all done, it should be possible to rebuild GDAL, and have the new format available in all the utilities. The *gdalinfo* utility can be used to test that opening and reporting on the format is working, and the *gdal_translate* utility can be used to test image reading.

10.1.2.6 Adding Georeferencing

Now we will take the example a step forward, adding georeferencing support. We add the following two virtual method overrides to JDEMDataset, taking care to exactly match the signature of the method on the GDALDataset base class.

```
CPLErr GetGeoTransform( double * padfTransform ) override;
const OGRSpatialReference* GetSpatialRef() const override;
```

The implementation of *GDALDataset::GetGeoTransform()* just copies the usual geotransform matrix into the supplied buffer. Note that *GDALDataset::GetGeoTransform()* may be called a lot, so it isn't generally wise to do a lot of computation in it. In many cases the Open() will collect the geotransform, and this method will just copy it over. Also note that the geotransform return is based on an anchor point at the top left corner of the top left pixel, not the center of pixel approach used in some packages.

```
CPLErr JDEMDataset::GetGeoTransform( double *padfTransform )
{
    const char *psHeader = reinterpret_cast<const char *>(m_abyHeader);
```

(continues on next page)

(continued from previous page)

```
    const double dfLLLat = JDEMGetAngle(psHeader + 29);
    const double dfLLLong = JDEMGetAngle(psHeader + 36);
    const double dfURLat = JDEMGetAngle(psHeader + 43);
    const double dfURLong = JDEMGetAngle(psHeader + 50);

    padfTransform[0] = dfLLLong;
    padfTransform[3] = dfURLat;
    padfTransform[1] = (dfURLong - dfLLLong) / GetRasterXSize();
    padfTransform[2] = 0.0;

    padfTransform[4] = 0.0;
    padfTransform[5] = -1 * (dfURLat - dfLLLat) / GetRasterYSize();

    return CE_None;
}
```

The *GDALDataset::GetSpatialRef()* method returns a pointer to an internal OGRSpatialReference object.

```
const OGRSpatialReference *JDEMDataset::GetSpatialRef() const
{
    return &m_oSRS;
}
```

In this case the coordinate system is fixed for all files of this format, and has been initialized in the JDEMDataset constructor. But in more complex cases, a definition may need to be composed on the fly, in which case it may be helpful to use the *OGRSpatialReference* class to help build the definition.

```
JDEMDataset::JDEMDataset()
{
    std::fill_n(m_abyHeader, CPL_ARRAYSIZE(m_abyHeader), static_cast<GByte>(0));
    m_oSRS.SetAxisMappingStrategy(OAMS_TRADITIONAL_GIS_ORDER);
    m_oSRS.importFromEPSG(4301); // Tokyo geographic CRS
}
```

This completes explanation of the features of the JDEM driver. The full source for jdemdataset.cpp can be reviewed as needed.

10.1.2.7 Overviews

GDAL allows file formats to make pre-built overviews available to applications via the *GDALRasterBand::GetOverview()* and related methods. However, implementing this is pretty involved, and goes beyond the scope of this document for now. The GeoTIFF driver (gdal/frmts/gtiff/geotiff.cpp) and related source can be reviewed for an example of a file format implementing overview reporting and creation support.

Formats can also report that they have arbitrary overviews, by overriding the *GDALRasterBand::HasArbitraryOverviews()* method on the GDALRasterBand, returning TRUE. In this case the raster band object is expected to override the *GDALRasterBand::RasterIO()* method itself, to implement efficient access to imagery with resampling. This is also involved, and there are a lot of requirements for correct implementation of the RasterIO() method. An example of this can be found in the OGDI and ECW formats.

However, by far the most common approach to implementing overviews is to use the default support in GDAL for external overviews stored in TIFF files with the same name as the dataset, but the extension .ovr appended. In order to enable reading and creation of this style of overviews it is necessary for the GDALDataset to initialize the *oOvManager*

object within itself. This is typically accomplished with a call like the following near the end of the Open() method (after the PAM `GDALDataset::TryLoadXML()`).

```
poDS->oOvManager.Initialize(poDS.get(), poOpenInfo->pszFilename);
```

This will enable default implementations for reading and creating overviews for the format. It is advised that this be enabled for all simple file system based formats unless there is a custom overview mechanism to be tied into.

10.1.2.8 File Creation

There are two approaches to file creation. The first method is called the *`GDALDriver::CreateCopy()`* method, and involves implementing a function that can write a file in the output format, pulling all imagery and other information needed from a source GDALDataset. The second method, the dynamic creation method, involves implementing a Create method to create the shell of the file, and then the application writes various information by calls to set methods.

The benefits of the first method are that that all the information is available at the point the output file is being created. This can be especially important when implementing file formats using external libraries which require information like color maps, and georeferencing information at the point the file is created. The other advantage of this method is that the CreateCopy() method can read some kinds of information, such as min/max, scaling, description and GCPs for which there are no equivalent set methods.

The benefits of the second method are that applications can create an empty new file, and write results to it as they become available. A complete image of the desired data does not have to be available in advance.

For very important formats both methods may be implemented, otherwise do whichever is simpler, or provides the required capabilities.

CreateCopy

The GDALDriver::CreateCopy() method call is passed through directly, so that method should be consulted for details of arguments. However, some things to keep in mind are:

- If the *bStrict* flag is FALSE the driver should try to do something reasonable when it cannot exactly represent the source dataset, transforming data types on the fly, dropping georeferencing and so forth.
- Implementing progress reporting correctly is somewhat involved. The return result of the progress function needs always to be checked for cancellation, and progress should be reported at reasonable intervals. The JPEGCreateCopy() method demonstrates good handling of the progress function.
- Special creation options should be documented in the on-line help. If the options take the format "NAME=VALUE" the papszOptions list can be manipulated with `CPLFetchNameValue()` as demonstrated in the handling of the QUALITY and PROGRESSIVE flags for JPEGCreateCopy().
- The returned GDALDataset handle can be in ReadOnly or Update mode. Return it in Update mode if practical, otherwise in ReadOnly mode is fine.

The full implementation of the CreateCopy function for JPEG (which is assigned to pfnCreateCopy in the GDALDriver object) is here. static GDALDataset *

```
JPEGCreateCopy( const char * pszFilename, GDALDataset *poSrcDS,
                int bStrict, char ** papszOptions,
                GDALProgressFunc pfnProgress, void * pProgressData )
{
    const int nBands = poSrcDS->GetRasterCount();
    const int nXSize = poSrcDS->GetRasterXSize();
    const int nYSize = poSrcDS->GetRasterYSize();
    // Some some rudimentary checks
    if( nBands != 1 && nBands != 3 )
```

(continues on next page)

(continued from previous page)

```
    {
        CPLError(CE_Failure, CPLE_NotSupported,
                "JPEG driver doesn't support %d bands.  Must be 1 (grey) "
                "or 3 (RGB) bands.", nBands);
        return NULL;
    }

    if( poSrcDS->GetRasterBand(1)->GetRasterDataType() != GDT_UInt8 && bStrict )
    {
        CPLError(CE_Failure, CPLE_NotSupported,
                "JPEG driver doesn't support data type %s. "
                "Only eight bit byte bands supported.",
                GDALGetDataTypeName(
                    poSrcDS->GetRasterBand(1)->GetRasterDataType()));
        return NULL;
    }

    // What options has the user selected?
    int nQuality = 75;
    if( CSLFetchNameValue(papszOptions, "QUALITY") != NULL )
    {
        nQuality = atoi(CSLFetchNameValue(papszOptions, "QUALITY"));
        if( nQuality < 10 || nQuality > 100 )
        {
            CPLError(CE_Failure, CPLE_IllegalArg,
                    "QUALITY=%s is not a legal value in the range 10 - 100.",
                    CSLFetchNameValue(papszOptions, "QUALITY"));
            return NULL;
        }
    }

    bool bProgressive = false;
    if( CSLFetchNameValue(papszOptions, "PROGRESSIVE") != NULL )
    {
        bProgressive = true;
    }

    // Create the dataset.
    VSILFILE *fpImage = VSIFOpenL(pszFilename, "wb");
    if( fpImage == NULL )
    {
        CPLError(CE_Failure, CPLE_OpenFailed,
                "Unable to create jpeg file %s.",
                pszFilename);
        return NULL;
    }

    // Initialize JPG access to the file.
    struct jpeg_compress_struct sCInfo;
    struct jpeg_error_mgr sJErr;
    sCInfo.err = jpeg_std_error(&sJErr);
    jpeg_create_compress(&sCInfo);
```

(continues on next page)

(continued from previous page)

```
    jpeg_stdio_dest(&sCInfo, fpImage);
    sCInfo.image_width = nXSize;
    sCInfo.image_height = nYSize;
    sCInfo.input_components = nBands;
    if( nBands == 1 )
    {
        sCInfo.in_color_space = JCS_GRAYSCALE;
    }
    else
    {
        sCInfo.in_color_space = JCS_RGB;
    }
    jpeg_set_defaults(&sCInfo);
    jpeg_set_quality(&sCInfo, nQuality, TRUE);
    if( bProgressive )
        jpeg_simple_progression(&sCInfo);
    jpeg_start_compress(&sCInfo, TRUE);

    // Loop over image, copying image data.
    GByte *pabyScanline = static_cast<GByte *>(CPLMalloc(nBands * nXSize));
    for( int iLine = 0; iLine < nYSize; iLine++ )
    {
        for( int iBand = 0; iBand < nBands; iBand++ )
        {
            GDALRasterBand * poBand = poSrcDS->GetRasterBand(iBand + 1);
            const CPLErr eErr =
                poBand->RasterIO(GF_Read, 0, iLine, nXSize, 1,
                                 pabyScanline + iBand, nXSize, 1, GDT_UInt8,
                                 nBands, nBands * nXSize);
            // TODO: Handle error.
        }
        JSAMPLE *ppSamples = pabyScanline;
        jpeg_write_scanlines(&sCInfo, &ppSamples, 1);
    }
    CPLFree(pabyScanline);
    jpeg_finish_compress(&sCInfo);
    jpeg_destroy_compress(&sCInfo);
    VSIFCloseL(fpImage);
    return static_cast<GDALDataset *>(GDALOpen(pszFilename, GA_ReadOnly));
}
```

Dynamic Creation

In the case of dynamic creation, there is no source dataset. Instead the size, number of bands, and pixel data type of the desired file is provided but other information (such as georeferencing, and imagery data) would be supplied later via other method calls on the resulting GDALDataset.

The following sample implement PCI .aux labeled raw raster creation. It follows a common approach of creating a blank, but valid file using non-GDAL calls, and then calling GDALOpen(,GA_Update) at the end to return a writable file handle. This avoids having to duplicate the various setup actions in the Open() function.

```
GDALDataset *PAuxDataset::Create( const char * pszFilename,
```

(continues on next page)

(continued from previous page)

```
                                int nXSize, int nYSize, int nBands,
                                GDALDataType eType,
                                char ** /* papszParamList */ )
{
    // Verify input options.
    if( eType != GDT_UInt8 && eType != GDT_Float32 &&
        eType != GDT_UInt16 && eType != GDT_Int16 )
    {
        CPLError(
            CE_Failure, CPLE_AppDefined,
            "Attempt to create PCI .Aux labeled dataset with an illegal "
            "data type (%s).",
            GDALGetDataTypeName(eType));
        return NULL;
    }

    // Try to create the file.
    FILE *fp = VSIFOpen(pszFilename, "w");
    if( fp == NULL )
    {
        CPLError(CE_Failure, CPLE_OpenFailed,
                "Attempt to create file `%s' failed.",
                pszFilename);
        return NULL;
    }

    // Just write out a couple of bytes to establish the binary
    // file, and then close it.
    VSIFWrite("\0\0", 2, 1, fp);
    VSIFClose(fp);

    // Create the aux filename.
    char *pszAuxFilename = static_cast<char *>(CPLMalloc(strlen(pszFilename) + 5));
    strcpy(pszAuxFilename, pszFilename);;
    for( int i = strlen(pszAuxFilename) - 1; i > 0; i-- )
    {
        if( pszAuxFilename[i] == '.' )
        {
            pszAuxFilename[i] = '\0';
            break;
        }
    }
    strcat(pszAuxFilename, ".aux");

    // Open the file.
    fp = VSIFOpen(pszAuxFilename, "wt");
    if( fp == NULL )
    {
        CPLError(CE_Failure, CPLE_OpenFailed,
                "Attempt to create file `%s' failed.",
                pszAuxFilename);
        return NULL;
```

(continues on next page)

(continued from previous page)

```
    }

    // We need to write out the original filename but without any
    // path components in the AuxiliaryTarget line.  Do so now.
    int iStart = strlen(pszFilename) - 1;
    while( iStart > 0 && pszFilename[iStart - 1] != '/' &&
        pszFilename[iStart - 1] != '\\' )
        iStart--;
    VSIFPrintf(fp, "AuxilaryTarget: %s\n", pszFilename + iStart);

    // Write out the raw definition for the dataset as a whole.
    VSIFPrintf(fp, "RawDefinition: %d %d %d\n",
            nXSize, nYSize, nBands);

    // Write out a definition for each band.  We always write band
    // sequential files for now as these are pretty efficiently
    // handled by GDAL.
    int nImgOffset = 0;
    for( int iBand = 0; iBand < nBands; iBand++ )
    {
        const int nPixelOffset = GDALGetDataTypeSize(eType)/8;
        const int nLineOffset = nXSize * nPixelOffset;
        const char *pszTypeName = NULL;
        if( eType == GDT_Float32 )
            pszTypeName = "32R";
        else if( eType == GDT_Int16 )
            pszTypeName = "16S";
        else if( eType == GDT_UInt16 )
            pszTypeName = "16U";
        else
            pszTypeName = "8U";
        VSIFPrintf( fp, "ChanDefinition-%d: %s %d %d %d %s\n",
                    iBand + 1, pszTypeName,
                    nImgOffset, nPixelOffset, nLineOffset,
#ifdef CPL_LSB
                    "Swapped"
#else
                    "Unswapped"
#endif
                    );
        nImgOffset += nYSize * nLineOffset;
    }

    // Cleanup.
    VSIFClose(fp);
    return static_cast<GDALDataset *>(GDALOpen(pszFilename, GA_Update));
}
```

File formats supporting dynamic creation, or even just update-in-place access also need to implement an IWriteBlock() method on the raster band class. It has semantics similar to IReadBlock(). As well, for various esoteric reasons, it is critical that a FlushCache() method be implemented in the raster band destructor. This is to ensure that any write cache blocks for the band be flushed out before the destructor is called.

10.1.2.9 RawDataset/RawRasterBand Helper Classes

Many file formats have the actual imagery data stored in a regular, binary, scanline oriented format. Rather than re-implement the access semantics for this for each formats, there are provided `RawDataset` and `RawRasterBand` classes declared in gcore/ that can be utilized to implement efficient and convenient access.

In these cases the format specific band class may not be required, or if required it can be derived from RawRasterBand. The dataset class should be derived from RawDataset.

The Open() method for the dataset then instantiates raster bands passing all the layout information to the constructor. For instance, the PNM driver uses the following calls to create it's raster bands.

```
if( poOpenInfo->pabyHeader[1] == '5' )
{
    poDS->SetBand(
        1, new RawRasterBand(poDS, 1, poDS->fpImage,
                             iIn, 1, nWidth, GDT_UInt8, TRUE));
}
else
{
    poDS->SetBand(
        1, new RawRasterBand(poDS, 1, poDS->fpImage,
                             iIn, 3, nWidth*3, GDT_UInt8, TRUE));
    poDS->SetBand(
        2, new RawRasterBand(poDS, 2, poDS->fpImage,
                             iIn+1, 3, nWidth*3, GDT_UInt8, TRUE));
    poDS->SetBand(
        3, new RawRasterBand(poDS, 3, poDS->fpImage,
                             iIn+2, 3, nWidth*3, GDT_UInt8, TRUE));
}
```

The RawRasterBand takes the following arguments.

- poDS: The GDALDataset this band will be a child of. This dataset must be of a class derived from RawRasterDataset.
- nBand: The band it is on that dataset, 1 based.
- fpRaw: The FILE * handle to the file containing the raster data.
- nImgOffset: The byte offset to the first pixel of raster data for the first scanline.
- nPixelOffset: The byte offset from the start of one pixel to the start of the next within the scanline.
- nLineOffset: The byte offset from the start of one scanline to the start of the next.
- eDataType: The GDALDataType code for the type of the data on disk.
- bNativeOrder: FALSE if the data is not in the same endianness as the machine GDAL is running on. The data will be automatically byte swapped.

Simple file formats utilizing the Raw services are normally placed all within one file in the gdal/frmts/raw directory. There are numerous examples there of format implementation.

10.1.2.10 Metadata, and Other Exotic Extensions

There are various other items in the GDAL data model, for which virtual methods exist on the GDALDataset and GDALRasterBand. They include:

- Metadata: Name/value text values about a dataset or band. The GDALMajorObject (base class for GDALRasterBand and GDALDataset) has built-in support for holding metadata, so for read access it only needs to be set with

calls to SetMetadataItem() during the Open(). The SAR_CEOS (frmts/ceos2/sar_ceosdataset.cpp) and GeoTIFF drivers are examples of drivers implementing readable metadata.

- ColorTables: GDT_UInt8 raster bands can have color tables associated with them. The frmts/png/pngdataset.cpp driver contains an example of a format that supports colortables.
- ColorInterpretation: The PNG driver contains an example of a driver that returns an indication of whether a band should be treated as a Red, Green, Blue, Alpha or Greyscale band.
- GCPs: GDALDatasets can have a set of ground control points associated with them (as opposed to an explicit affine transform returned by GetGeotransform()) relating the raster to georeferenced coordinates. The MFF2 (gdal/frmts/raw/hkvdataset.cpp) format is a simple example of a format supporting GCPs.
- NoDataValue: Bands with known "nodata" values can implement the GetNoDataValue() method. See the PAux (frmts/raw/pauxdataset.cpp) for an example of this.
- Category Names: Classified images with names for each class can return them using the GetCategoryNames() method though no formats currently implement this.

10.1.3 GDAL Warp API tutorial (Reprojection, ...)

10.1.3.1 Overview

The GDAL Warp API (declared in *gdalwarper.h*) provides services for high performance image warping using application provided geometric transformation functions (GDALTransformerFunc), a variety of resampling kernels, and various masking options. Files much larger than can be held in memory can be warped.

This tutorial demonstrates how to implement an application using the Warp API. It assumes implementation in C++ as C and Python bindings are incomplete for the Warp API. It also assumes familiarity with the *Raster Data Model*, and the general GDAL API.

Applications normally perform a warp by initializing a `GDALWarpOptions` structure with the options to be utilized, instantiating a GDALWarpOperation based on these options, and then invoking the `GDALWarpOperation::ChunkAndWarpImage()` method to perform the warp options internally using the `GDALWarpKernel` class.

10.1.3.2 A Simple Reprojection Case

First we will construct a relatively simple example for reprojecting an image, assuming an appropriate output file already exists, and with minimal error checking.

```
#include "gdalwarper.h"

int main()
{
    GDALDatasetH  hSrcDS, hDstDS;

    // Open input and output files.
    GDALAllRegister();
    hSrcDS = GDALOpen( "in.tif", GA_ReadOnly );
    hDstDS = GDALOpen( "out.tif", GA_Update );

    // Setup warp options.
    GDALWarpOptions *psWarpOptions = GDALCreateWarpOptions();
    psWarpOptions->hSrcDS = hSrcDS;
    psWarpOptions->hDstDS = hDstDS;
    psWarpOptions->nBandCount = 1;
    psWarpOptions->panSrcBands =
```

(continues on next page)

(continued from previous page)

```
        (int *) CPLMalloc(sizeof(int) * psWarpOptions->nBandCount );
    psWarpOptions->panSrcBands[0] = 1;
    psWarpOptions->panDstBands =
        (int *) CPLMalloc(sizeof(int) * psWarpOptions->nBandCount );
    psWarpOptions->panDstBands[0] = 1;
    psWarpOptions->pfnProgress = GDALTermProgress;

    // Establish reprojection transformer.
    psWarpOptions->pTransformerArg =
        GDALCreateGenImgProjTransformer( hSrcDS,
                                         GDALGetProjectionRef(hSrcDS),
                                         hDstDS,
                                         GDALGetProjectionRef(hDstDS),
                                         FALSE, 0.0, 1 );
    psWarpOptions->pfnTransformer = GDALGenImgProjTransform;

    // Initialize and execute the warp operation.
    GDALWarpOperation oOperation;
    oOperation.Initialize( psWarpOptions );
    oOperation.ChunkAndWarpImage( 0, 0,
                                  GDALGetRasterXSize( hDstDS ),
                                  GDALGetRasterYSize( hDstDS ) );
    GDALDestroyGenImgProjTransformer( psWarpOptions->pTransformerArg );
    GDALDestroyWarpOptions( psWarpOptions );
    GDALClose( hDstDS );
    GDALClose( hSrcDS );
    return 0;
}
```

This example opens the existing input and output files (in.tif and out.tif). A *GDALWarpOptions* structure is allocated (*GDALCreateWarpOptions()* sets lots of sensible defaults for stuff, always use it for defaulting things), and the input and output file handles, and band lists are set. The panSrcBands and panDstBands lists are dynamically allocated here and will be free automatically by *GDALDestroyWarpOptions()*. The simple terminal output progress monitor (GDALTermProgress) is installed for reporting completion progress to the user.

GDALCreateGenImgProjTransformer() is used to initialize the reprojection transformation between the source and destination images. We assume that they already have reasonable bounds and coordinate systems set. Use of GCPs is disabled.

Once the options structure is ready, a GDALWarpOperation is instantiated using them, and the warp actually performed with *GDALWarpOperation::ChunkAndWarpImage()*. Then the transformer, warp options and datasets are cleaned up.

Normally error check would be needed after opening files, setting up the reprojection transformer (returns NULL on failure), and initializing the warp.

10.1.3.3 Other Warping Options

The GDALWarpOptions structures contains a number of items that can be set to control warping behavior. A few of particular interest are:

- *GDALWarpOptions::dfWarpMemoryLimit* - Set the maximum amount of memory to be used by the GDALWarpOperation when selecting a size of image chunk to operate on. The value is in bytes, and the default is likely to be conservative (small). Increasing the chunk size can help substantially in some situations but care should be taken to ensure that this size, plus the GDAL cache size plus the working set of GDAL, your application and the

operating system are less than the size of RAM or else excessive swapping is likely to interfere with performance. On a system with 256MB of RAM, a value of at least 64MB (roughly 64000000 bytes) is reasonable. Note that this value does not include the memory used by GDAL for low level block caching.

- *GDALWarpOptions::eResampleAlg* - One of GRA_NearestNeighbour (the default, and fastest), GRA_Bilinear (2x2 bilinear resampling) or GRA_Cubic. The GRA_NearestNeighbour type should generally be used for thematic or color mapped images. The other resampling types may give better results for thematic images, especially when substantially changing resolution.
- *GDALWarpOptions::padfSrcNoDataReal* - This array (one entry per band being processed) may be setup with a "nodata" value for each band if you wish to avoid having pixels of some background value copied to the destination image.
- *GDALWarpOptions::papszWarpOptions* - This is a string list of NAME=VALUE options passed to the warper. See the *GDALWarpOptions::papszWarpOptions* docs for all options. Supported values include:
  - INIT_DEST=[value] or INIT_DEST=NO_DATA: This option forces the destination image to be initialized to the indicated value (for all bands) or indicates that it should be initialized to the NO_DATA value in padfDstNoDataReal/padfDstNoDataImag. If this value isn't set the destination image will be read and the source warp overlaid on it.
  - WRITE_FLUSH=YES/NO: This option forces a flush to disk of data after each chunk is processed. In some cases this helps ensure a serial writing of the output data otherwise a block of data may be written to disk each time a block of data is read for the input buffer resulting in a lot of extra seeking around the disk, and reduced IO throughput. The default at this time is NO.

10.1.3.4 Creating the Output File

In the previous case an appropriate output file was already assumed to exist. Now we will go through a case where a new file with appropriate bounds in a new coordinate system is created. This operation doesn't relate specifically to the warp API. It is just using the transformation API.

```
#include "gdalwarper.h"
#include "ogr_spatialref.h"
...
GDALDriverH hDriver;
GDALDataType eDT;
GDALDatasetH hDstDS;
GDALDatasetH hSrcDS;

// Open the source file.
hSrcDS = GDALOpen( "in.tif", GA_ReadOnly );
CPLAssert( hSrcDS != NULL );

// Create output with same datatype as first input band.
eDT = GDALGetRasterDataType(GDALGetRasterBand(hSrcDS,1));

// Get output driver (GeoTIFF format)
hDriver = GDALGetDriverByName( "GTiff" );
CPLAssert( hDriver != NULL );

// Get Source coordinate system.
const char *pszSrcWKT, *pszDstWKT = NULL;
pszSrcWKT = GDALGetProjectionRef( hSrcDS );
CPLAssert( pszSrcWKT != NULL && strlen(pszSrcWKT) > 0 );
```

(continues on next page)

(continued from previous page)

```
// Setup output coordinate system that is UTM 11 WGS84.
OGRSpatialReference oSRS;
oSRS.SetUTM( 11, TRUE );
oSRS.SetWellKnownGeogCS( "WGS84" );
oSRS.exportToWkt( &pszDstWKT );

// Create a transformer that maps from source pixel/line coordinates
// to destination georeferenced coordinates (not destination
// pixel line).  We do that by omitting the destination dataset
// handle (setting it to NULL).
void *hTransformArg;
hTransformArg =
    GDALCreateGenImgProjTransformer( hSrcDS, pszSrcWKT, NULL, pszDstWKT,
                                     FALSE, 0, 1 );
CPLAssert( hTransformArg != NULL );

// Get approximate output georeferenced bounds and resolution for file.
double adfDstGeoTransform[6];
int nPixels=0, nLines=0;
CPLErr eErr;
eErr = GDALSuggestedWarpOutput( hSrcDS,
                                GDALGenImgProjTransform, hTransformArg,
                                adfDstGeoTransform, &nPixels, &nLines );
CPLAssert( eErr == CE_None );
GDALDestroyGenImgProjTransformer( hTransformArg );

// Create the output file.
hDstDS = GDALCreate( hDriver, "out.tif", nPixels, nLines,
                     GDALGetRasterCount(hSrcDS), eDT, NULL );
CPLAssert( hDstDS != NULL );

// Write out the projection definition.
GDALSetProjection( hDstDS, pszDstWKT );
GDALSetGeoTransform( hDstDS, adfDstGeoTransform );

// Copy the color table, if required.
GDALColorTableH hCT;
hCT = GDALGetRasterColorTable( GDALGetRasterBand(hSrcDS,1) );
if( hCT != NULL )
    GDALSetRasterColorTable( GDALGetRasterBand(hDstDS,1), hCT );
... proceed with warp as before ...
```

Some notes on this logic:

- We need to create the transformer to output coordinates such that the output of the transformer is georeferenced, not pixel line coordinates since we use the transformer to map pixels around the source image into destination georeferenced coordinates.
- The *GDALSuggestedWarpOutput()* function will return an adfDstGeoTransform, nPixels and nLines that describes an output image size and georeferenced extents that should hold all pixels from the source image. The resolution is intended to be comparable to the source, but the output pixels are always square regardless of the shape of input pixels.
- The warper requires an output file in a format that can be "randomly" written to. This generally limits things to

uncompressed formats that have an implementation of the Create() method (as opposed to CreateCopy()). To warp to compressed formats, or CreateCopy() style formats it is necessary to produce a full temporary copy of the image in a better behaved format, and then CreateCopy() it to the desired final format.

- The Warp API copies only pixels. All color maps, georeferencing and other metadata must be copied to the destination by the application.

10.1.3.5 Performance Optimization

There are a number of things that can be done to optimize the performance of the warp API:

- Increase the amount of memory available for the Warp API chunking so that larger chunks can be operated on at a time. This is the *GDALWarpOptions::dfWarpMemoryLimit* parameter. In theory the larger the chunk size operated on the more efficient the I/O strategy, and the more efficient the approximated transformation will be. However, the sum of the warp memory and the GDAL cache should be less than RAM size, likely around 2/3 of RAM size.
- Increase the amount of memory for GDAL caching. This is especially important when working with very large input and output images that are scanline oriented. If all the input or output scanlines have to be re-read for each chunk they intersect performance may degrade greatly. Use *GDALSetCacheMax()* to control the amount of memory available for caching within GDAL.
- Use an approximated transformation instead of exact reprojection for each pixel to be transformed. This code illustrates how an approximated transformation could be created based on a reprojection transformation, but with a given error threshold (dfErrorThreshold in output pixels).

```
hTransformArg =
    GDALCreateApproxTransformer( GDALGenImgProjTransform,
                                 hGenImgProjArg, dfErrorThreshold );
pfnTransformer = GDALApproxTransform;
```

- When writing to a blank output file, use the INIT_DEST option in the *GDALWarpOptions::papszWarpOptions* to cause the output chunks to be initialized to a fixed value, instead of being read from the output. This can substantially reduce unnecessary IO work.
- Use tiled input and output formats. Tiled formats allow a given chunk of source and destination imagery to be accessed without having to touch a great deal of extra image data. Large scanline oriented files can result in a great deal of wasted extra IO.
- Process all bands in one call. This ensures the transformation calculations don't have to be performed for each band.
- Use the *GDALWarpOperation::ChunkAndWarpMulti()* method instead of *GDALWarpOperation::ChunkAndWarpImage()*. It uses a separate thread for the IO and the actual image warp operation allowing more effective use of CPU and IO bandwidth. For this to work GDAL needs to have been built with multi-threading support (default on Win32, default on Unix, for previous versions -with-threads was required in configure).
- The resampling kernels vary in work required from nearest neighbour being least complex, to more complex with bilinear and then cubic. Don't use a more complex resampling kernel than needed.
- Avoid use of esoteric masking options so that special simplified logic case be used for common special cases. For instance, nearest neighbour resampling with no masking on 8bit data is highly optimized compared to the general case.

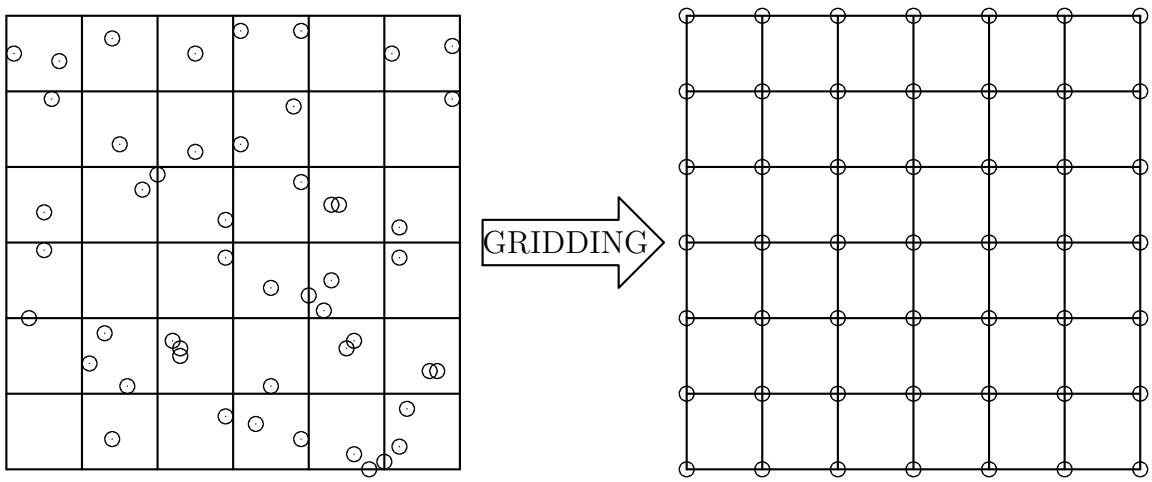
10.1.3.6 Other Masking Options

The GDALWarpOptions include a bunch of esoteric masking capabilities, for validity masks, and density masks on input and output. Some of these are not yet implemented and others are implemented but poorly tested. Other than per-band validity masks it is advised that these features be used with caution at this time.

10.1.4 GDAL Grid Tutorial

10.1.4.1 Introduction to Gridding

Gridding is a process of creating a regular grid (or call it a raster image) from the scattered data. Typically you have a set of arbitrary data scattered over the region of survey measurements and you would like to convert them into the regular grid for further processing and combining with other grids.



This problem can be solved using data interpolation or approximation algorithms. But you are not limited by interpolation here. Sometimes you don't need to interpolate your data but rather compute some statistics or data metrics over the region. Statistics are valuable themselves or could be used for better choosing the interpolation algorithm and parameters.

That is what GDAL Grid API is about. It helps you to interpolate your data (see *Interpolation of the Scattered Data*) or compute data metrics (see *Data Metrics Computation*).

There are two ways of using this interface. Programmatically it is available through the `GDALGridCreate()` C function; for end users there is a *gdal_grid* utility (which see for an up-to-date list of all current features). The rest of this document discusses details on algorithms and their parameters implemented in GDAL Grid API.

10.1.4.2 Interpolation of the Scattered Data

Inverse Distance to a Power

The Inverse Distance to a Power gridding method is a weighted average interpolator. You should supply the input arrays with the scattered data values including coordinates of every data point and output grid geometry. The function will compute an interpolated value for the given position in the output grid.

For every grid node the resulting value Z will be calculated using the formula:

$$Z = \frac{\sum_{i=1}^{n} \frac{Z_i}{r_i^p}}{\sum_{i=1}^{n} \frac{1}{r_i^p}}$$

where:

- Z_i is a known value at point i,

- r_i is a distance from the grid node to point i,
- p is a weighting power,
- n is a number of points in *Search Ellipse*.

The smoothing parameter s is used as an additive term in the Euclidean distance calculation:

$$r_i = \sqrt{{r_{ix}}^2 + {r_{iy}}^2 + s^2}$$

where r_{ix} and r_{iy} are the horizontal and vertical distances between the grid node to point i respectively.

In this method the weighting factor w is

$$w = \frac{1}{r^p}$$

See `GDALGridInverseDistanceToAPowerOptions` for the list of `GDALGridCreate()` parameters and *invdist* for the list of *gdal_grid* options.

Moving Average

The Moving Average is a simple data averaging algorithm. It uses a moving window of elliptic form to search values and averages all data points within the window. *Search Ellipse* can be rotated by specified angle, the center of ellipse located at the grid node. Also the minimum number of data points to average can be set. If there are not enough points in window, the grid node is considered empty and will be filled with specified NODATA value.

Mathematically it can be expressed with the formula:

$$Z = \frac{\sum_{i=1}^{n} Z_i}{n}$$

where:

- Z is a resulting value at the grid node,
- Z_i is a known value at point i,
- n is a number of points in search *Search Ellipse*.

See `GDALGridMovingAverageOptions` for the list of `GDALGridCreate()` parameters and *average* for the list of *gdal_grid* options.

Nearest Neighbor

The Nearest Neighbor method doesn't perform any interpolation or smoothing, it just takes the value of nearest point found in the grid node search ellipse and returns it as a result. If there are no points found, the specified NODATA value will be returned.

See `GDALGridNearestNeighborOptions` for the list of `GDALGridCreate()` parameters and *nearest* for the list of *gdal_grid* options.

10.1.4.3 Data Metrics Computation

All the metrics have the same set controlling options. See the `GDALGridDataMetricsOptions`.

Minimum Data Value

Minimum value found in grid node *Search Ellipse*. If there are no points found, the specified NODATA value will be returned.

$$Z = \min\left(Z_1, Z_2, \ldots, Z_n\right)$$

where:

- Z is a resulting value at the grid node,
- Z_i is a known value at point i,
- n is a number of points in *Search Ellipse*.

Maximum Data Value

Maximum value found in grid node *Search Ellipse*. If there are no points found, the specified NODATA value will be returned.

$$Z = \max\left(Z_1, Z_2, \ldots, Z_n\right)$$

where:

- Z is a resulting value at the grid node,
- Z_i is a known value at point i,
- n is a number of points in *Search Ellipse*.

Data Range

A difference between the minimum and maximum values found in grid *Search Ellipse*. If there are no points found, the specified NODATA value will be returned.

$$Z = \max\left(Z_1, Z_2, \ldots, Z_n\right) - \min\left(Z_1, Z_2, \ldots, Z_n\right)$$

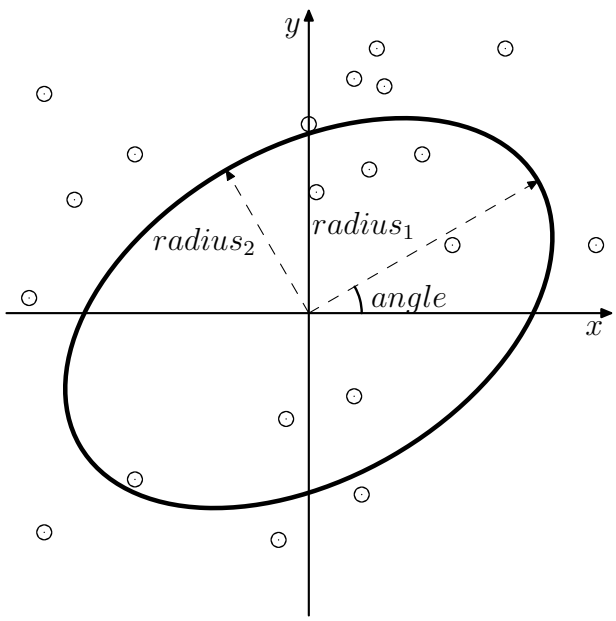
where:

- Z is a resulting value at the grid node,
- Z_i is a known value at point i,
- n is a number of points in *Search Ellipse*.

10.1.4.4 Search Ellipse

Search window in gridding algorithms specified in the form of rotated ellipse. It is described by the three parameters:

- $radius_1$ is the first radius (x axis if rotation angle is 0),
- $radius_2$ is the second radius (y axis if rotation angle is 0),
- $angle$ is a search ellipse rotation angle (rotated counter clockwise).



Only points located inside the search ellipse (including its border line) will be used for computation.

10.1.5 Geotransform Tutorial

10.1.5.1 Introduction to Geotransforms:

A geotransform is an affine transformation from the image coordinate space (row, column), also known as (pixel, line) to the georeferenced coordinate space (projected or geographic coordinates).

A geotransform consists in a set of 6 coefficients:

`GT(0)` x-coordinate of the upper-left corner of the upper-left pixel.
`GT(1)` w-e pixel resolution / pixel width.
`GT(2)` row rotation (typically zero).
`GT(3)` y-coordinate of the upper-left corner of the upper-left pixel.
`GT(4)` column rotation (typically zero).
`GT(5)` n-s pixel resolution / pixel height (negative value for a north-up image).

10.1.5.2 Transformation from image coordinate space to georeferenced coordinate space:

```
X_geo = GT(0) + X_pixel * GT(1) + Y_line * GT(2)
Y_geo = GT(3) + X_pixel * GT(4) + Y_line * GT(5)
```

Note that the pixel/line coordinates in the above are from (0.0,0.0) at the top left corner of the top left pixel to (width_in_pixels,height_in_pixels) at the bottom right corner of the bottom right pixel. The pixel/line location of the center of the top left pixel would therefore be (0.5,0.5).

10.1.5.3 In case of north up images:

`GT(2)`, `GT(4)` coefficients are zero.
`GT(1)`, `GT(5)` is the pixel size.
`GT(0)`, `GT(3)` position is the top left corner of the top left pixel of the raster.

10.1.6 Working with a Digital Terrain Model (DTM)

Contents

- *Working with a Digital Terrain Model (DTM)*
  - *Overview*
  - *Getting the Data*
  - *Creating a Raster Tile Index*
  - *Clipping the DTM*
  - *Processing the DTM*

10.1.6.1 Overview

This tutorial demonstrates how to work with a DTM (Digital Terrain Model), using freely available datasets. You'll learn how to create a raster tile index, clip data to a polygon, and generate hillshade, aspect, and slope images with various GDAL commands.

Code examples assume a Bash shell on Linux. Where syntax differs, equivalent PowerShell commands are provided for Windows users. The tutorial covers both the unified *gdal command-line interface* and the *traditional* individual GDAL tools. The tutorial uses a Conda environment with GDAL installed, see *Conda setup* for setting this up.

10.1.6.2 Getting the Data

We'll use open data from IGN (Institut national de l'information géographique et forestière), France's national mapping agency. IGN provides a national DTM (the RGEALTI dataset) available at https://geoservices.ign.fr/rgealti.

This tutorial focuses on department 90 - "Territoire de Belfort" (a department is an administrative unit in France). The DTM for this area is available as a `.7z` archive: RGEALTI_2-0_5M_ASC_LAMB93-IGN69_D090_2021-01-13.7z (45 MB). You can automate the download and extraction using the commands below.

Bash

PowerShell

```
# ensure 7-zip is installed
apt-get update && apt-get install curl p7zip-full -y

# use curl to download the file
curl -O https://data.geopf.fr/telechargement/download/RGEALTI/RGEALTI_2-0_5M_ASC_LAMB93-
↪IGN69_D090_2021-01-13/RGEALTI_2-0_5M_ASC_LAMB93-IGN69_D090_2021-01-13.7z

# unzip the contents
7z x RGEALTI_2-0_5M_ASC_LAMB93-IGN69_D090_2021-01-13.7z
```

```
# hiding progress output significantly speeds up downloading files
$ProgressPreference = "SilentlyContinue"

# download the zip file
Invoke-WebRequest `
  -Uri "https://data.geopf.fr/telechargement/download/RGEALTI/RGEALTI_2-0_5M_ASC_LAMB93-
↪IGN69_D090_2021-01-13/RGEALTI_2-0_5M_ASC_LAMB93-IGN69_D090_2021-01-13.7z" `
```

(continues on next page)

(continued from previous page)

```
  -OutFile "RGEALTI_2-0_5M_ASC_LAMB93-IGN69_D090_2021-01-13.7z"

# download the 7-zip installer
Invoke-WebRequest -Uri "https://www.7-zip.com/a/7z2409-x64.exe" -OutFile "7z-x64.exe"

# unzip the contents
Start-Process -FilePath "7z-x64.exe" -ArgumentList "/S" -Wait
& "C:\Program Files\7-Zip\7z.exe" x "RGEALTI_2-0_5M_ASC_LAMB93-IGN69_D090_2021-01-13.7z"␣
↪-y
```

10.1.6.3 Creating a Raster Tile Index

The archive contains individual tiles in the ASCII raster (`.asc`) format - a plain text format commonly used for gridded spatial data. The following commands rely on the *AAIGrid -- Arc/Info ASCII Grid* driver being installed (this is typically included in standard GDAL builds).

RGEALTI_FXX_0980_6730_MNT_LAMB93_IGN69.asc	ASC File	6,838 KB
RGEALTI_FXX_0980_6735_MNT_LAMB93_IGN69.asc	ASC File	6,838 KB
RGEALTI_FXX_0980_6740_MNT_LAMB93_IGN69.asc	ASC File	6,838 KB
RGEALTI_FXX_0980_6745_MNT_LAMB93_IGN69.asc	ASC File	6,838 KB
RGEALTI_FXX_0980_6750_MNT_LAMB93_IGN69.asc	ASC File	6,943 KB
RGEALTI_FXX_0980_6755_MNT_LAMB93_IGN69.asc	ASC File	7,116 KB

Rather than working with each file separately, we can treat them as a single virtual raster dataset using the *GDAL Raster Tile Index* driver. The tile index can be created using the new CLI tool *gdal driver gti create* or with *gdaltindex* from the traditional GDAL tools. This creates a vector dataset with one record per raster tile, allowing the tiles to be accessed as a unified raster in GDAL, MapServer, or QGIS (version 3.4+).

gdal CLI

Traditional

PowerShell

```
files="RGEALTI_2-0_5M_ASC_LAMB93-IGN69_D090_2021-01-13/RGEALTI/1_DONNEES_LIVRAISON_2021-
↪10-00009/RGEALTI_MNT_5M_ASC_LAMB93_IGN69_D090/*.asc"
gdal driver gti create --layer dtm --dst-crs EPSG:2154 $files tileindex.gti.fgb
```

```
files="RGEALTI_2-0_5M_ASC_LAMB93-IGN69_D090_2021-01-13/RGEALTI/1_DONNEES_LIVRAISON_2021-
↪10-00009/RGEALTI_MNT_5M_ASC_LAMB93_IGN69_D090/*.asc"
gdaltindex -lyr_name dtm -t_srs EPSG:2154 tileindex.gti.fgb $files
```

```
# create a GTI from all the asc files
$files = "RGEALTI_2-0_5M_ASC_LAMB93-IGN69_D090_2021-01-13/RGEALTI/1_DONNEES_LIVRAISON_
↪2021-10-00009/RGEALTI_MNT_5M_ASC_LAMB93_IGN69_D090/*.asc"
gdal driver gti create --layer dtm --dst-crs EPSG:2154 $files tileindex.gti.fgb
```

10.1.6.4 Clipping the DTM

Next, we'll clip the DTM using a polygon representing department 90. Instead of downloading a national dataset of all departments, we can use a WFS server to retrieve only the feature we need.

For the traditional approach we'll use *ogr2ogr* and the *WFS - OGC WFS service* driver. Starting with GDAL 3.11, the same result can be achieved using the GDAL CLI and the *gdal vector pipeline*. In both cases, your GDAL installation must include the the *GML - Geography Markup Language* driver.

We will explicitly set the output layer name to *commune* to avoid the auto-generated name *ADMINEXPRESS-COG.2017:commune*. When using `ogr2ogr` it's also important to set the geometry type with `-nlt POLYGON`, otherwise, clipping the raster will fail with the error: `ERROR 1: Cutline not of polygon type`.

gdal CLI

Traditional

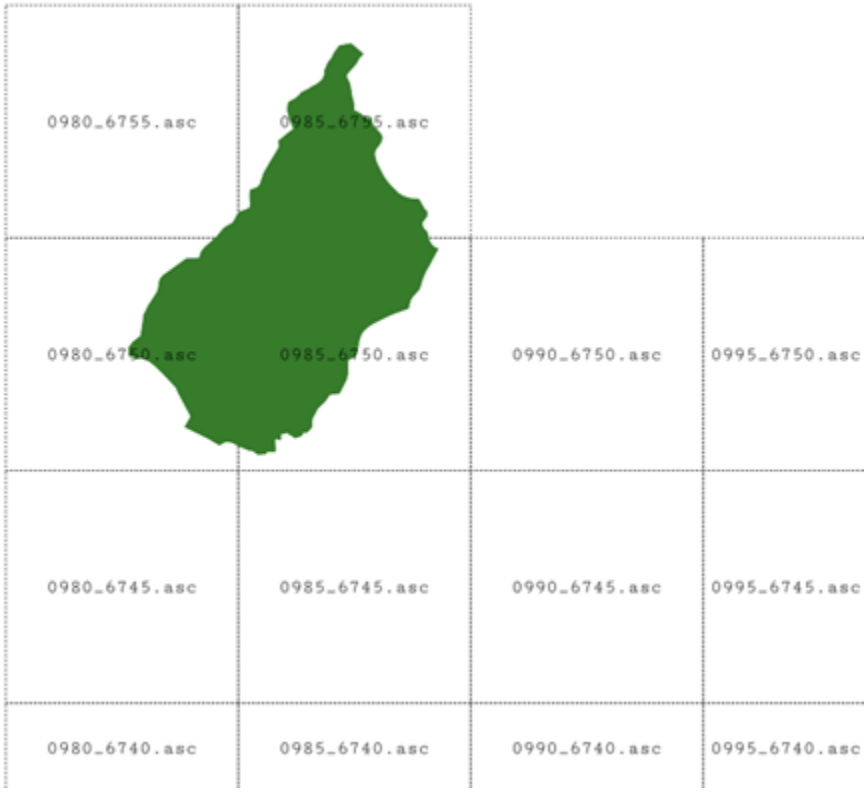
PowerShell

```
gdal vector pipeline \
    ! read WFS:"https://data.geopf.fr/wfs/ows?version=2.0.0&typename=ADMINEXPRESS-COG.
↪2017:commune" \
    ! set-geom-type --geometry-type Polygon \
    ! filter --where "insee_com='90065'" \
    ! write --output-layer=commune commune.fgb
```

```
ogr2ogr -f FlatGeobuf commune.fgb \
    WFS:"https://data.geopf.fr/wfs/ows?version=2.0.0&typename=ADMINEXPRESS-COG.
↪2017:commune" \
    -nln commune \
    -nlt POLYGON \
    -where "insee_com = '90065'"
```

```
gdal vector pipeline `
    ! read WFS:"https://data.geopf.fr/wfs/ows?version=2.0.0&typename=ADMINEXPRESS-COG.
↪2017:commune" `
    ! set-geom-type --geometry-type Polygon `
    ! filter --where "insee_com='90065'" `
    ! write --output-layer=commune commune.fgb
```

The image below shows the polygon downloaded from the WFS server overlaid on the raster tile index. We will clip the tiled raster using this polygon to create a new **clipped.tif** file using *gdal raster clip* for the CLI approach, and *gdalwarp* for the traditional method.



gdal CLI

Traditional

```
gdal raster clip --like commune.fgb  tileindex.gti.fgb clipped.tif --overwrite
```

```
gdalwarp -cutline commune.fgb -crop_to_cutline -of GTiff tileindex.gti.fgb clipped.tif
```

Note

To perform spatial operations such as clipping, your GDAL installation must be built with the GEOS (Geometry Engine - Open Source) dependency. You can verify this using the GDAL Python library provided in the GDAL Conda environment.

```
# if GEOS is installed the following will output the GEOS version number
$ python -c "from osgeo import ogr; print(ogr.GetGEOSVersionMajor())"
3
```

If GEOS is not installed, you may encounter errors like the following:

```
# ERROR 6: GEOS support not enabled.
```

10.1.6.5 Processing the DTM

With a raster covering our area of interest, we can now use GDAL's powerful raster tools. For an excellent overview of creating relief maps with GDAL, see A Gentle Introduction to GDAL Part 5: Shaded Relief.

The traditional tools for analyzing and visualizing DEM s and DTM s are the *gdaldem* commands. The new CLI provides similar functionality under the `gdal raster` subcommands.

First we'll create a shaded relief map using *gdaldem* and `hillshade` (traditional approach) or *gdal raster hillshade* with the GDAL CLI. Both commands support various parameters, such as light azimuth, altitude, and vertical exaggeration. In this example, we'll set the vertical exaggeration factor to `2` (using the `-z 2` and `--zfactor=2` options) to enhance the shading effect.

Refer to the command help pages for full details, and feel free to experiment with the examples below.

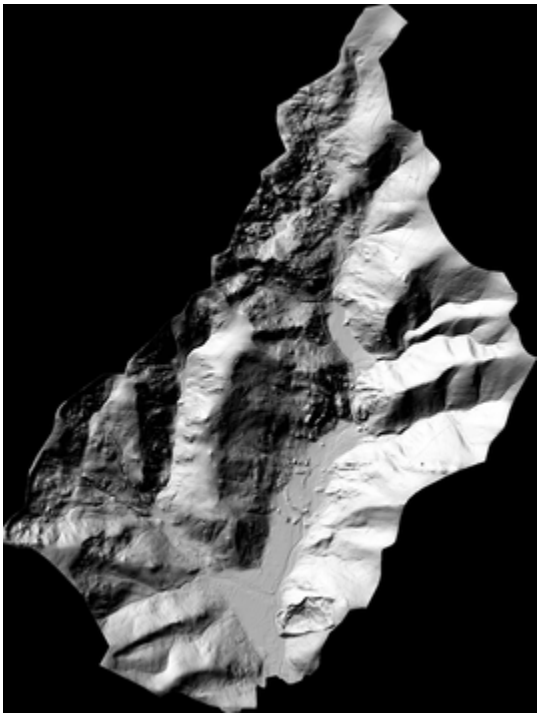
gdal CLI

Traditional

```
gdal raster hillshade --zfactor=2 clipped.tif hillshade.tif --overwrite
```

```
gdaldem hillshade -z 2 clipped.tif hillshade.tif
```

The output of this command is shown below. Note that images have been converted to the PNG format and resized for the GDAL documentation.



Next, we'll create a slope map using *gdaldem* and `slope` (traditional approach), or *gdal raster slope* with the GDAL CLI. Feel free to experiment with the available settings to produce a visualization that best represents the data.

gdal CLI

Traditional

```
gdal raster slope clipped.tif slope.tif --overwrite
```

```
gdaldem slope clipped.tif slope.tif
```

Now we'll create an image showing terrain aspect using *gdaldem* and `aspect` (traditional approach) or *gdal raster aspect* with the GDAL CLI.

gdal CLI

Traditional

```
gdal raster aspect clipped.tif aspect.tif --overwrite
```

```
gdaldem aspect clipped.tif aspect.tif
```

Next, we'll apply color to the DTM based on elevation values. First, create a file named `color-map-percentage.txt` in the same folder as your `clipped.tif` file, and copy in the contents below. The file defines RGB values for elevation ranges, grouped by percentage. GDAL maps these percentage ranges to RGB colors when rendering the DTM. The `nv` (no data value) entry specifies the RGB colour for any `NoData` values in the raster. In this case it is set to `255 255 255 0`, where the final `0` means the no data areas will be fully transparent.

```
nv 255 255 255 0
0% 112 147 141
10% 120 159 152
20% 130 165 159
30% 145 177 171
40% 180 192 180
50% 212 201 180
60% 212 184 163
70% 212 193 179
80% 212 207 204
90% 220 220 220
100% 235 235 237
```

Now we can apply the color map to the DTM using *gdaldem* and `color-relief` (traditional approach) or *gdal raster color-map* with the GDAL CLI. To make the background transparent, include the `-alpha` (traditional) or `--add-alpha` (CLI) option when running the command.

gdal CLI

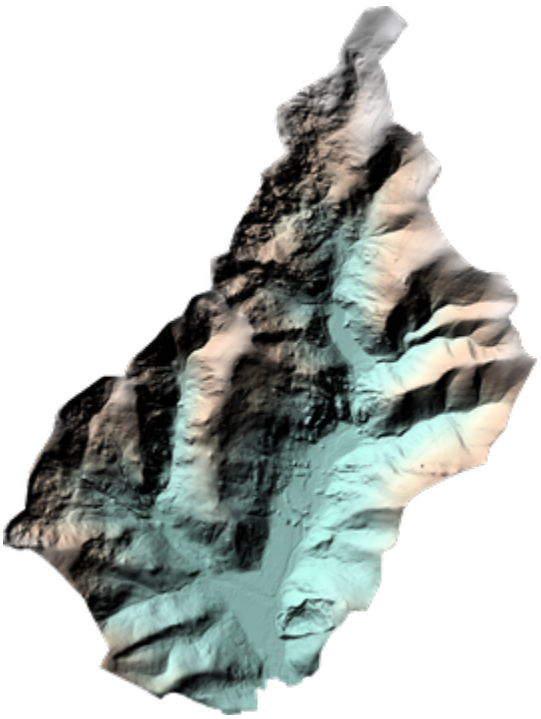
Traditional

```
gdal raster color-map --color-map=color-map-percentage.txt clipped.tif color.tif --add-
↪alpha --overwrite
```

```
gdaldem color-relief clipped.tif color-map-percentage.txt color.tif -alpha
```

To finish, we'll use a new tool introduced in GDAL 3.12 - *gdal raster blend* - to blend the color map with the previously created hillshade image. This command is available only through the new GDAL CLI.

```
gdal raster blend --operator=hsv-value --grayscale=hillshade.tif color.tif color-
↪hillshade.png --overwrite
```



10.2 Multidimensional raster

10.2.1 Multidimensional raster API tutorial

10.2.1.1 Read the content of an array

C++

C

Python

```
// Note: since GDAL >= 3.12, "gdal_multidim_cpp.h" can also be used.
#include "gdal_priv.h"

int main()
{
    GDALAllRegister();
    auto poDataset = std::unique_ptr<GDALDataset>(
        GDALDataset::Open( "in.nc", GDAL_OF_MULTIDIM_RASTER ));
    if( !poDataset )
    {
        exit(1);
    }
    auto poRootGroup = poDataset->GetRootGroup();
    if( !poRootGroup )
    {
        exit(1);
    }
    auto poVar = poRootGroup->OpenMDArray("temperature");
    if( !poVar )
```

(continues on next page)

(continued from previous page)

```
    {
        exit(1);
    }
    size_t nValues = 1;
    std::vector<size_t> anCount;
    for( const auto poDim: poVar->GetDimensions() )
    {
        anCount.push_back(static_cast<size_t>(poDim->GetSize()));
        nValues *= anCount.back();
    }
    std::vector<double> values(nValues);
    poVar->Read(std::vector<GUInt64>{0,0,0}.data(),
                anCount.data(),
                nullptr, /* step: defaults to 1,1,1 */
                nullptr, /* stride: default to row-major convention */
                GDALExtendedDataType::Create(GDT_Float64),
                &values[0]);
    return 0;
}
```

```
#include "gdal.h"
#include "cpl_conv.h"
int main()
{
    GDALDatasetH hDS;
    GDALGroupH hGroup;
    GDALMDArrayH hVar;
    size_t nDimCount;
    GDALDimensionH* dims;
    size_t nValues;
    size_t i;
    size_t* panCount;
    GUInt64* panOffset;
    double* padfValues;
    GDALExtendedDataTypeH hDT;

    GDALAllRegister();
    hDS = GDALOpenEx( "in.nc", GDAL_OF_MULTIDIM_RASTER, NULL, NULL, NULL);
    if( !hDS )
    {
        exit(1);
    }
    hGroup = GDALDatasetGetRootGroup(hDS);
    GDALReleaseDataset(hDS);
    if( !hGroup )
    {
        exit(1);
    }
    hVar = GDALGroupOpenMDArray(hGroup, "temperature", NULL);
    GDALGroupRelease(hGroup);
    if( !hVar )
    {
```

(continues on next page)

(continued from previous page)

```
        exit(1);
    }

    dims = GDALMDArrayGetDimensions(hVar, &nDimCount);
    panCount = (size_t*)CPLMalloc(nDimCount * sizeof(size_t));
    nValues = 1;
    for( i = 0; i < nDimCount; i++ )
    {
        panCount[i] = GDALDimensionGetSize(dims[i]);
        nValues *= panCount[i];
    }
    GDALReleaseDimensions(dims, nDimCount);
    panOffset = (GUInt64*)CPLCalloc(nDimCount, sizeof(GUInt64));

    padfValues = (double*)VSIMalloc2(nValues, sizeof(double));
    if( !padfValues )
    {
        GDALMDArrayRelease(hVar);
        CPLFree(panOffset);
        CPLFree(panCount);
        exit(1);
    }
    hDT = GDALExtendedDataTypeCreate(GDT_Float64);
    GDALMDArrayRead(hVar,
                    panOffset,
                    panCount,
                    NULL, /* step: defaults to 1,1,1 */
                    NULL, /* stride: default to row-major convention */
                    hDT,
                    padfValues,
                    NULL, /* array start. Omitted */
                    0 /* array size in bytes. Omitted */);
    GDALExtendedDataTypeRelease(hDT);
    GDALMDArrayRelease(hVar);
    CPLFree(panOffset);
    CPLFree(panCount);
    VSIFree(padfValues);

    return 0;
}
```

```
from osgeo import gdal
ds = gdal.OpenEx("in.nc", gdal.OF_MULTIDIM_RASTER)
rootGroup = ds.GetRootGroup()
var = rootGroup.OpenMDArray("temperature")
data = var.Read(buffer_datatype = gdal.ExtendedDataType.Create(gdal.GDT_Float64))

# if NumPy is available we can use ReadAsArray
data = var.ReadAsArray(buffer_datatype = gdal.ExtendedDataType.Create(gdal.GDT_Float64))
```

10.2.1.2 Other examples

Test scripts from the GDAL autotest suite

- *MEM* driver: memmultidim.py
- *netCDF* driver: netcdf_multidim.py
- *HDF5* driver: hdf5multidim.py
- *VRT* driver: vrtmultidim.py
- *gdalmdiminfo* tests: test_gdalmdiminfo_lib.py
- *gdalmdimtranslate* tests: test_gdalmdimtranslate_lib.py

10.3 Vector

10.3.1 Vector API tutorial

This document is intended to document using the OGR C++ classes to read and write data from a file. It is strongly advised that the reader first review the *Vector Data Model* document describing the key classes and their roles in OGR.

It also includes code snippets for the corresponding functions in C and Python.

10.3.1.1 Reading From OGR

For purposes of demonstrating reading with OGR, we will construct a small utility for dumping point layers from an OGR data source to stdout in comma-delimited format.

Initially it is necessary to register all the format drivers that are desired. This is normally accomplished by calling `GDALAllRegister()` which registers all format drivers built into GDAL/OGR.

C++

C

Python

```
// Note: since GDAL >= 3.12, "gdal_vector_cpp.h" can also be used.
#include "ogrsf_frmts.h"

int main()

{
    GDALAllRegister();
```

```
#include "gdal.h"

int main()

{
    GDALAllRegister();
```

```
from osgeo import gdal
# when importing gdal in Python
# GDALAllRegister() is automatically called
```

Next we need to open the input OGR datasource. Datasources can be files, RDBMSes, directories full of files, or even remote web services depending on the driver being used. However, the datasource name is always a single string. In this case we are hardcoded to open a particular shapefile. The second argument (GDAL_OF_VECTOR) tells the *OGROpen()* method that we want a vector driver to be use and that don't require update access. On failure NULL is returned, and we report an error.

C++

C

Python

```
GDALDataset       *poDS;

poDS = (GDALDataset*) GDALOpenEx( "point.shp", GDAL_OF_VECTOR, NULL, NULL, NULL );
if( poDS == NULL )
{
    printf( "Open failed.\n" );
    exit( 1 );
}
```

```
GDALDatasetH hDS;

hDS = GDALOpenEx( "point.shp", GDAL_OF_VECTOR, NULL, NULL, NULL );
if( hDS == NULL )
{
    printf( "Open failed.\n" );
    exit( 1 );
}
```

```
ds = gdal.OpenEx("point", gdal.OF_VECTOR)
```

A GDALDataset can potentially have many layers associated with it. The number of layers available can be queried with *GDALDataset::GetLayerCount()* and individual layers fetched by index using *GDALDataset::GetLayer()*. However, we will just fetch the layer by name.

C++

C

Python

```
OGRLayer  *poLayer;

poLayer = poDS->GetLayerByName( "point" );
```

```
OGRLayerH hLayer;

hLayer = GDALDatasetGetLayerByName( hDS, "point" );
```

```
lyr = ds.GetLayerByName("point")
```

Now we want to start reading features from the layer. Before we start we could assign an attribute or spatial filter to the layer to restrict the set of feature we get back, but for now we are interested in getting all features.

C++

C

Python

```
for( auto& poFeature: poLayer )
{
      // do something with each feature
}
```

```
OGR_FOR_EACH_FEATURE_BEGIN(hFeature, hLayer)
{
     // do something, including continue, break;
     // do not explicitly destroy the feature (unless you use return or goto
     // outside of the loop, in which case use OGR_F_Destroy(hFeat))
}
OGR_FOR_EACH_FEATURE_END(hFeat)
```

```
for feat in lyr:
  # do something with each feature
```

In order to dump all the attribute fields of the feature, it is helpful to get the *OGRFeatureDefn*. This is an object, associated with the layer, containing the definitions of all the fields. We loop over all the fields, and fetch and report the attributes based on their type.

C++

C

Python

```
for( auto&& oField: *poFeature )
{
    if( oField.IsUnset() )
    {
        printf("(unset),");
        continue;
    }
    if( oField.IsNull() )
    {
        printf("(null),");
        continue;
    }
    switch( oField.GetType() )
    {
        case OFTInteger:
            printf( "%d,", oField.GetInteger() );
            break;
        case OFTInteger64:
            printf( CPL_FRMT_GIB ",", oField.GetInteger64() );
            break;
        case OFTReal:
            printf( "%.3f,", oField.GetDouble() );
            break;
        case OFTString:
            // GetString() returns a C string
```

(continues on next page)

(continued from previous page)

```
            printf( "%s,", oField.GetString() );
            break;
        default:
            // Note: we use GetAsString() and not GetString(), since
            // the later assumes the field type to be OFTString while the
            // former will do a conversion from the original type to string.
            printf( "%s,", oField.GetAsString() );
            break;
    }
}
```

```
OGRFeatureDefnH hFDefn = OGR_L_GetLayerDefn(hLayer);
int iField;

for( iField = 0; iField < OGR_FD_GetFieldCount(hFDefn); iField++ )
{
    OGRFieldDefnH hFieldDefn = OGR_FD_GetFieldDefn( hFDefn, iField );

    if( !OGR_F_IsFieldSet(hFeature, iField) )
    {
        printf("(unset),");
        continue;
    }
    if( OGR_F_IsFieldNull(hFeature, iField) )
    {
        printf("(null),");
        continue;
    }

    switch( OGR_Fld_GetType(hFieldDefn) )
    {
        case OFTInteger:
            printf( "%d,", OGR_F_GetFieldAsInteger( hFeature, iField ) );
            break;
        case OFTInteger64:
            printf( CPL_FRMT_GIB ",", OGR_F_GetFieldAsInteger64( hFeature, iField ) );
            break;
        case OFTReal:
            printf( "%.3f,", OGR_F_GetFieldAsDouble( hFeature, iField) );
            break;
        case OFTString:
            printf( "%s,", OGR_F_GetFieldAsString( hFeature, iField) );
            break;
        default:
            printf( "%s,", OGR_F_GetFieldAsString( hFeature, iField) );
            break;
    }
}
```

```
feat_defn = lyr.GetLayerDefn()
for i in range(feat_defn.GetFieldCount()):
```

(continues on next page)

(continued from previous page)

```
    field_defn = feat_defn.GetFieldDefn(i)

    # Tests below can be simplified with just :
    # print feat.GetField(i)
    if (
        field_defn.GetType() == ogr.OFTInteger
        or field_defn.GetType() == ogr.OFTInteger64
    ):
        print("%d" % feat.GetFieldAsInteger64(i))
    elif field_defn.GetType() == ogr.OFTReal:
        print("%.3f" % feat.GetFieldAsDouble(i))
    elif field_defn.GetType() == ogr.OFTString:
        print("%s" % feat.GetFieldAsString(i))
    else:
        print("%s" % feat.GetFieldAsString(i))
```

There are a few more field types than those explicitly handled above, but a reasonable representation of them can be fetched with the *OGRFeature::GetFieldAsString()* method. In fact we could shorten the above by using GetFieldAsString() for all the types.

Next we want to extract the geometry from the feature, and write out the point geometry x and y. Geometries are returned as a generic *OGRGeometry* pointer. We then determine the specific geometry type, and if it is a point, we cast it to point and operate on it. If it is something else we write placeholders.

C++

C

Python

```
OGRGeometry *poGeometry;

poGeometry = poFeature->GetGeometryRef();
if( poGeometry != NULL
        && wkbFlatten(poGeometry->getGeometryType()) == wkbPoint )
{
#if GDAL_VERSION_NUM >= GDAL_COMPUTE_VERSION(2,3,0)
    OGRPoint *poPoint = poGeometry->toPoint();
#else
    OGRPoint *poPoint = (OGRPoint *) poGeometry;
#endif

    printf( "%.3f,%3.f\n", poPoint->getX(), poPoint->getY() );
}
else
{
    printf( "no point geometry\n" );
}
```

```
OGRGeometryH hGeometry;

hGeometry = OGR_F_GetGeometryRef(hFeature);
if( hGeometry != NULL
        && wkbFlatten(OGR_G_GetGeometryType(hGeometry)) == wkbPoint )
```

(continues on next page)

(continued from previous page)

```
{
    printf( "%.3f,%3.f\n", OGR_G_GetX(hGeometry, 0), OGR_G_GetY(hGeometry, 0) );
}
else
{
    printf( "no point geometry\n" );
}
```

```
geom = feat.GetGeometryRef()
if geom is not None and geom.GetGeometryType() == ogr.wkbPoint:
    print("%.3f, %.3f" % (geom.GetX(), geom.GetY()))
else:
    print("no point geometry")
```

The `wkbFlatten()` macro is used above to convert the type for a wkbPoint25D (a point with a z coordinate) into the base 2D geometry type code (wkbPoint). For each 2D geometry type there is a corresponding 2.5D type code. The 2D and 2.5D geometry cases are handled by the same C++ class, so our code will handle 2D or 3D cases properly.

Several geometry fields can be associated to a feature.

C++

C

Python

```
OGRGeometry *poGeometry;
int iGeomField;
int nGeomFieldCount;

nGeomFieldCount = poFeature->GetGeomFieldCount();
for(iGeomField = 0; iGeomField < nGeomFieldCount; iGeomField ++ )
{
    poGeometry = poFeature->GetGeomFieldRef(iGeomField);
    if( poGeometry != NULL
            && wkbFlatten(poGeometry->getGeometryType()) == wkbPoint )
    {
#if GDAL_VERSION_NUM >= GDAL_COMPUTE_VERSION(2,3,0)
        OGRPoint *poPoint = poGeometry->toPoint();
#else
        OGRPoint *poPoint = (OGRPoint *) poGeometry;
#endif

        printf( "%.3f,%3.f\n", poPoint->getX(), poPoint->getY() );
    }
    else
    {
        printf( "no point geometry\n" );
    }
}
```

```
OGRGeometryH hGeometry;
int iGeomField;
int nGeomFieldCount;
```

(continues on next page)

(continued from previous page)

```
nGeomFieldCount = OGR_F_GetGeomFieldCount(hFeature);
for(iGeomField = 0; iGeomField < nGeomFieldCount; iGeomField ++ )
{
    hGeometry = OGR_F_GetGeomFieldRef(hFeature, iGeomField);
    if( hGeometry != NULL
            && wkbFlatten(OGR_G_GetGeometryType(hGeometry)) == wkbPoint )
    {
        printf( "%.3f,%3.f\n", OGR_G_GetX(hGeometry, 0),
                OGR_G_GetY(hGeometry, 0) );
    }
    else
    {
        printf( "no point geometry\n" );
    }
}
```

```
nGeomFieldCount = feat.GetGeomFieldCount()
for iGeomField in range(nGeomFieldCount):
    geom = feat.GetGeomFieldRef(iGeomField)
    if geom is not None and geom.GetGeometryType() == ogr.wkbPoint:
        print("%.3f, %.3f" % ( geom.GetX(), geom.GetY() ))
    else:
        print("no point geometry\n")
```

Note that *OGRFeature::GetGeometryRef()* and *OGRFeature::GetGeomFieldRef()* return a pointer to the internal geometry owned by the OGRFeature. There we don't actually delete the return geometry.

The OGRLayer returned by *GDALDataset::GetLayerByName()* is also a reference to an internal layer owned by the GDALDataset so we don't need to delete it. But we do need to delete the datasource in order to close the input file. Once again we do this with a custom delete method to avoid special win32 heap issues.

C++

C

Python

```
GDALClose( poDS );
}
```

```
GDALClose( poDS );
}
```

```
ds.Close()
```

All together our program looks like this.

C++

C

Python

```
#include "ogrsf_frmts.h"

int main()

{
    GDALAllRegister();

    GDALDatasetUniquePtr poDS(GDALDataset::Open( "point.shp", GDAL_OF_VECTOR));
    if( poDS == nullptr )
    {
        printf( "Open failed.\n" );
        exit( 1 );
    }

    for( const OGRLayer* poLayer: poDS->GetLayers() )
    {
        for( const auto& poFeature: *poLayer )
        {
            for( const auto& oField: *poFeature )
            {
                if( oField.IsUnset() )
                {
                    printf("(unset),");
                    continue;
                }
                if( oField.IsNull() )
                {
                    printf("(null),");
                    continue;
                }
                switch( oField.GetType() )
                {
                    case OFTInteger:
                        printf( "%d,", oField.GetInteger() );
                        break;
                    case OFTInteger64:
                        printf( CPL_FRMT_GIB ",", oField.GetInteger64() );
                        break;
                    case OFTReal:
                        printf( "%.3f,", oField.GetDouble() );
                        break;
                    case OFTString:
                        // GetString() returns a C string
                        printf( "%s,", oField.GetString() );
                        break;
                    default:
                        // Note: we use GetAsString() and not GetString(), since
                        // the later assumes the field type to be OFTString while the
                        // former will do a conversion from the original type to string.
                        printf( "%s,", oField.GetAsString() );
                        break;
                }
            }
```

(continues on next page)

(continued from previous page)

```
            const OGRGeometry *poGeometry = poFeature->GetGeometryRef();
            if( poGeometry != nullptr
                    && wkbFlatten(poGeometry->getGeometryType()) == wkbPoint )
            {
                const OGRPoint *poPoint = poGeometry->toPoint();

                printf( "%.3f,%3.f\n", poPoint->getX(), poPoint->getY() );
            }
            else
            {
                printf( "no point geometry\n" );
            }
        }
    }
    return 0;
}
```

```
#include "gdal.h"

int main()

{
    GDALAllRegister();

    GDALDatasetH hDS;
    OGRLayerH hLayer;
    OGRFeatureH hFeature;
    OGRFeatureDefnH hFDefn;

    hDS = GDALOpenEx( "point.shp", GDAL_OF_VECTOR, NULL, NULL, NULL );
    if( hDS == NULL )
    {
        printf( "Open failed.\n" );
        exit( 1 );
    }

    hLayer = GDALDatasetGetLayerByName( hDS, "point" );
    hFDefn = OGR_L_GetLayerDefn(hLayer);

    OGR_L_ResetReading(hLayer);
    while( (hFeature = OGR_L_GetNextFeature(hLayer)) != NULL )
    {
        int iField;
        OGRGeometryH hGeometry;

        for( iField = 0; iField < OGR_FD_GetFieldCount(hFDefn); iField++ )
        {
            OGRFieldDefnH hFieldDefn = OGR_FD_GetFieldDefn( hFDefn, iField );

            if( !OGR_F_IsFieldSet(hFeature, iField) )
            {
```

(continues on next page)

(continued from previous page)

```
                printf("(unset),");
                continue;
            }
            if( OGR_F_IsFieldNull(hFeature, iField) )
            {
                printf("(null),");
                continue;
            }

            switch( OGR_Fld_GetType(hFieldDefn) )
            {
                case OFTInteger:
                    printf( "%d,", OGR_F_GetFieldAsInteger( hFeature, iField ) );
                    break;
                case OFTInteger64:
                    printf( CPL_FRMT_GIB ",", OGR_F_GetFieldAsInteger64( hFeature,␣
↪iField ) );
                    break;
                case OFTReal:
                    printf( "%.3f,", OGR_F_GetFieldAsDouble( hFeature, iField) );
                    break;
                case OFTString:
                    printf( "%s,", OGR_F_GetFieldAsString( hFeature, iField) );
                    break;
                default:
                    printf( "%s,", OGR_F_GetFieldAsString( hFeature, iField) );
                    break;
            }
        }

        hGeometry = OGR_F_GetGeometryRef(hFeature);
        if( hGeometry != NULL
            && wkbFlatten(OGR_G_GetGeometryType(hGeometry)) == wkbPoint )
        {
            printf( "%.3f,%3.f\n", OGR_G_GetX(hGeometry, 0), OGR_G_GetY(hGeometry, 0) );
        }
        else
        {
            printf( "no point geometry\n" );
        }

        OGR_F_Destroy( hFeature );
    }

    GDALClose( hDS );
}
```

```
from osgeo import gdal, ogr

gdal.UseExceptions()

ds = gdal.OpenEx("point", gdal.OF_VECTOR)
```

(continues on next page)

(continued from previous page)

```
lyr = ds.GetLayerByName("point")

lyr.ResetReading()

for feat in lyr:

    feat_defn = lyr.GetLayerDefn()
    for i in range(feat_defn.GetFieldCount()):
        field_defn = feat_defn.GetFieldDefn(i)

        # Tests below can be simplified with just :
        # print feat.GetField(i)
        if (
            field_defn.GetType() == ogr.OFTInteger
            or field_defn.GetType() == ogr.OFTInteger64
        ):
            print("%d" % feat.GetFieldAsInteger64(i))
        elif field_defn.GetType() == ogr.OFTReal:
            print("%.3f" % feat.GetFieldAsDouble(i))
        elif field_defn.GetType() == ogr.OFTString:
            print("%s" % feat.GetFieldAsString(i))
        else:
            print("%s" % feat.GetFieldAsString(i))

    geom = feat.GetGeometryRef()
    if geom is not None and geom.GetGeometryType() == ogr.wkbPoint:
        print("%.3f, %.3f" % (geom.GetX(), geom.GetY()))
    else:
        print("no point geometry")

ds.Close()
```

10.3.1.2 Reading From OGR using the Arrow C Stream data interface

Added in version 3.6.

Instead of retrieving features one at a time, it is also possible to retrieve them by batches, with a column-oriented memory layout, using the *OGRLayer::GetArrowStream()* method. Note that this method is more difficult to use than the traditional *OGRLayer::GetNextFeature()* approach, and is only advised when compatibility with the Apache Arrow C Stream interface is needed, or when column-oriented consumption of layers is required.

Pending using an helper library, consumption of the Arrow C Stream interface requires reading of the following documents:

- Arrow C Stream interface
- Arrow C data interface
- Arrow Columnar Format.

The Arrow C Stream interface interface consists of a set of C structures, ArrowArrayStream, that provides two main callbacks to get:

- a ArrowSchema with the get_schema() callback. A ArrowSchema describes a set of field descriptions (name, type, metadata). All OGR data types have a corresponding Arrow data type.

- a sequence of ArrowArray with the get_next() callback. A ArrowArray captures a set of values for a specific column/field in a subset of features. This is the equivalent of a Series in a Pandas DataFrame. This is a potentially hierarchical structure that can aggregate sub arrays, and in OGR usage, the main array will be a StructArray which is the collection of OGR attribute and geometry fields. The layout of buffers and children arrays per data type is detailed in the Arrow Columnar Format.

If a layer consists of 4 features with 2 fields (one of integer type, one of floating-point type), the representation as a ArrowArray is *conceptually* the following one:

```
array.children[0].buffers[1] = { 1, 2, 3, 4 };
array.children[1].buffers[1] = { 1.2, 2.3, 3.4, 4.5 };
```

The content of a whole layer can be seen as a sequence of record batches, each record batches being an ArrowArray of a subset of features. Instead of iterating over individual features, one iterates over a batch of several features at once.

The ArrowArrayStream, ArrowSchema, ArrowArray structures are defined in a ogr_recordbatch.h public header file, directly derived from https://github.com/apache/arrow/blob/main/cpp/src/arrow/c/abi.h to get API/ABI compatibility with Apache Arrow C++. This header file must be explicitly included when the related array batch API is used.

The GetArrowStream() method has the following signature:

```
virtual bool OGRLayer::GetArrowStream(struct ArrowArrayStream* out_stream,
                                      CSLConstList papszOptions = nullptr);
```

It is also available in the C API as *OGR_L_GetArrowStream()*.

out_stream is a pointer to a ArrowArrayStream structure, that can be in a uninitialized state (the method will ignore any initial content).

On successful return, and when the stream interfaces is no longer needed, it must must be freed with out_stream->release(out_stream).

There are extra precautions to take into account in a OGR context. Unless otherwise specified by a particular driver implementation, the ArrowArrayStream structure, and the ArrowSchema or ArrowArray objects its callbacks have returned, should no longer be used (except for potentially being released) after the OGRLayer from which it was initialized has been destroyed (typically at dataset closing). Furthermore, unless otherwise specified by a particular driver implementation, only one ArrowArrayStream can be active at a time on a given layer (that is the last active one must be explicitly released before a next one is asked). Changing filter state, ignored columns, modifying the schema or using ResetReading()/GetNextFeature() while using a ArrowArrayStream is strongly discouraged and may lead to unexpected results. As a rule of thumb, no OGRLayer methods that affect the state of a layer should be called on a layer, while an ArrowArrayStream on it is active.

The papszOptions that may be provided is a NULL terminated list of key=value strings, that may be driver specific.

OGRLayer has a base implementation of GetArrowStream() that is such:

- The get_schema() callback returns a schema whose top-level object returned is of type Struct, and whose children consist of the FID column, all OGR attribute fields and geometry fields to Arrow fields. The FID column may be omitted by providing the INCLUDE_FID=NO option.

  When get_schema() returns 0, and the schema is no longer needed, it must be released with the following procedure, to take into account that it might have been released by other code, as documented in the Arrow C data interface:

  ```
  if( out_schema->release )
      out_schema->release(out_schema)
  ```

- The get_next() callback retrieve the next record batch over the layer.

out_array is a pointer to a ArrowArray structure, that can be in a uninitialized state (the method will ignore any initial content).

The default implementation uses GetNextFeature() internally to retrieve batches of up to 65,536 features (configurable with the MAX_FEATURES_IN_BATCH=num option). The starting address of buffers allocated by the default implementation is aligned on 64-byte boundaries.

The default implementation outputs geometries as WKB in a binary field, whose corresponding entry in the schema is marked with the metadata item `ARROW:extension:name` set to `ogc.wkb`. Specialized implementations may output by default other formats (particularly the Arrow driver that can return geometries encoded according to the GeoArrow specification (using a list of coordinates). The GEOMETRY_ENCODING=WKB option can be passed to force the use of WKB (through the default implementation)

The method may take into account ignored fields set with SetIgnoredFields() (the default implementation does), and should take into account filters set with SetSpatialFilter() and SetAttributeFilter(). Note however that specialized implementations may fallback to the default (slower) implementation when filters are set.

Mixing calls to GetNextFeature() and get_next() is not recommended, as the behavior will be unspecified (but it should not crash).

When get_next() returns 0, and the array is no longer needed, it must be released with the following procedure, to take into account that it might have been released by other code, as documented in the Arrow C data interface:

```
if( out_array->release )
    out_array->release(out_array)
```

Drivers that have a specialized implementation advertise the new OLCFastGetArrowStream layer capability.

Using directly (as a producer or a consumer) a ArrowArray is admittedly not trivial, and requires good intimacy with the Arrow C data interface and columnar array specifications, to know, in which buffer of an array, data is to be read, which data type void* buffers should be cast to, how to use buffers that contain null/not_null information, how to use offset buffers for data types of type List, etc. The study of the gdal_array._RecordBatchAsNumpy() method of the SWIG Python bindings (gdal_array.i) can give a good hint of how to use an ArrowArray object, in conjunction with the associated ArrowSchema.

The below example illustrates how to read the content of a layer that consists of a integer field and a geometry field:

```
#include "gdal_priv.h"
#include "ogr_api.h"
#include "ogrsf_frmts.h"
#include "ogr_recordbatch.h"
#include <cassert>

int main(int argc, char* argv[])
{
    GDALAllRegister();
    GDALDataset* poDS = GDALDataset::Open(argv[1]);
    if( poDS == nullptr )
    {
        CPLError(CE_Failure, CPLE_AppDefined, "Open() failed\n");
        exit(1);
    }
    OGRLayer* poLayer = poDS->GetLayer(0);
    OGRLayerH hLayer = OGRLayer::ToHandle(poLayer);

    // Get the Arrow stream
    struct ArrowArrayStream stream;
```

(continues on next page)

(continued from previous page)

```
    if( !OGR_L_GetArrowStream(hLayer, &stream, nullptr))
    {
        CPLError(CE_Failure, CPLE_AppDefined, "OGR_L_GetArrowStream() failed\n");
        delete poDS;
        exit(1);
    }

    // Get the schema
    struct ArrowSchema schema;
    if( stream.get_schema(&stream, &schema) != 0 )
    {
        CPLError(CE_Failure, CPLE_AppDefined, "get_schema() failed\n");
        stream.release(&stream);
        delete poDS;
        exit(1);
    }

    // Check that the returned schema consists of one int64 field (for FID),
    // one int32 field and one binary/wkb field
    if( schema.n_children != 3 ||
        strcmp(schema.children[0]->format, "l") != 0 || // int64 -> FID
        strcmp(schema.children[1]->format, "i") != 0 || // int32
        strcmp(schema.children[2]->format, "z") != 0 )  // binary for WKB
    {
        CPLError(CE_Failure, CPLE_AppDefined,
                 "Layer has not the expected schema required by this example.");
        schema.release(&schema);
        stream.release(&stream);
        delete poDS;
        exit(1);
    }
    schema.release(&schema);

    // Iterate over batches
    while( true )
    {
        struct ArrowArray array;
        if( stream.get_next(&stream, &array) != 0 ||
            array.release == nullptr )
        {
            break;
        }

        assert(array.n_children == 3);

        // Cast the array->children[].buffers[] to the appropriate data types
        const auto int_child = array.children[1];
        assert(int_child->n_buffers == 2);
        const uint8_t* int_field_not_null = static_cast<const uint8_t*>(int_child->
↪buffers[0]);
        const int32_t* int_field = static_cast<const int32_t*>(int_child->buffers[1]);
```

(continues on next page)

(continued from previous page)

```
        const auto wkb_child = array.children[2];
        assert(wkb_child->n_buffers == 3);
        const uint8_t* wkb_field_not_null = static_cast<const uint8_t*>(wkb_child->
↪buffers[0]);
        const int32_t* wkb_offset = static_cast<const int32_t*>(wkb_child->buffers[1]);
        const uint8_t* wkb_field = static_cast<const uint8_t*>(wkb_child->buffers[2]);

        // Lambda to check if a field is set for a given feature index
        const auto IsSet = [](const uint8_t* buffer_not_null, int i)
        {
            return buffer_not_null == nullptr || (buffer_not_null[i/8] >> (i%8)) != 0;
        };

        // Iterate through features of a batch
        for( long long i = 0; i < array.length; i++ )
        {
            if( IsSet(int_field_not_null, i) )
                printf("int_field[%lld] = %d\n", i, int_field[i]);
            else
                printf("int_field[%lld] = null\n", i);

            if( IsSet(wkb_field_not_null, i) )
            {
                const void* wkb = wkb_field + wkb_offset[i];
                const int32_t length = wkb_offset[i+1] - wkb_offset[i];
                char* wkt = nullptr;
                OGRGeometry* geom = nullptr;
                OGRGeometryFactory::createFromWkb(wkb, nullptr, &geom, length);
                if( geom )
                {
                    geom->exportToWkt(&wkt);
                }
                printf("wkb_field[%lld] = %s\n", i, wkt ? wkt : "invalid geometry");
                CPLFree(wkt);
                delete geom;
            }
            else
            {
                printf("wkb_field[%lld] = null\n", i);
            }
        }

        // Release memory taken by the batch
        array.release(&array);
    }

    // Release stream and dataset
    stream.release(&stream);
    delete poDS;
    return 0;
}
```

To write features by batches using an ArrowArray, consult *Writing to OGR using the Arrow C Data interface*.

10.3.1.3 Writing To OGR

As an example of writing through OGR, we will do roughly the opposite of the above. A short program that reads comma separated values from input text will be written to a point shapefile via OGR.

As usual, we start by registering all the drivers, and then fetch the Shapefile driver as we will need it to create our output file.

C++

C

Python

```
#include "ogrsf_frmts.h"

int main()
{
    const char *pszDriverName = "ESRI Shapefile";
    GDALDriver *poDriver;

    GDALAllRegister();

    poDriver = GetGDALDriverManager()->GetDriverByName(pszDriverName );
    if( poDriver == NULL )
    {
        printf( "%s driver not available.\n", pszDriverName );
        exit( 1 );
    }
```

```
#include "ogr_api.h"

int main()
{
    const char *pszDriverName = "ESRI Shapefile";
    GDALDriver *poDriver;

    GDALAllRegister();

    poDriver = (GDALDriver*) GDALGetDriverByName(pszDriverName );
    if( poDriver == NULL )
    {
        printf( "%s driver not available.\n", pszDriverName );
        exit( 1 );
    }
```

```
from osgeo import gdal

gdal.UseExceptions()
driverName = "ESRI Shapefile"
drv = gdal.GetDriverByName(driverName)
if drv is None:
    print("%s driver not available." % driverName)
    sys.exit(1)
```

Next we create the datasource. The ESRI Shapefile driver allows us to create a directory full of shapefiles, or a single

shapefile as a datasource. In this case we will explicitly create a single file by including the extension in the name. Other drivers behave differently. The second, third, fourth and fifth argument are related to raster dimensions (in case the driver has raster capabilities). The last argument to the call is a list of option values, but we will just be using defaults in this case. Details of the options supported are also format specific.

C++

C

Python

```
GDALDataset *poDS;

poDS = poDriver->Create( "point_out.shp", 0, 0, 0, GDT_Unknown, NULL );
if( poDS == NULL )
{
    printf( "Creation of output file failed.\n" );
    exit( 1 );
}
```

```
GDALDatasetH hDS;

hDS = GDALCreate( hDriver, "point_out.shp", 0, 0, 0, GDT_Unknown, NULL );
if( hDS == NULL )
{
    printf( "Creation of output file failed.\n" );
    exit( 1 );
}
```

```
ds = drv.Create("point_out.shp", 0, 0, 0, gdal.GDT_Unknown)
```

Now we create the output layer. In this case since the datasource is a single file, we can only have one layer. We pass wkbPoint to specify the type of geometry supported by this layer. In this case we aren't passing any coordinate system information or other special layer creation options.

C++

C

Python

```
OGRLayer *poLayer;

poLayer = poDS->CreateLayer( "point_out", NULL, wkbPoint, NULL );
if( poLayer == NULL )
{
    printf( "Layer creation failed.\n" );
    exit( 1 );
}
```

```
OGRLayerH hLayer;

hLayer = GDALDatasetCreateLayer( hDS, "point_out", NULL, wkbPoint, NULL );
if( hLayer == NULL )
{
    printf( "Layer creation failed.\n" );
```

(continues on next page)

(continued from previous page)

```
    exit( 1 );
}
```

```
lyr = ds.CreateLayer("point_out", None, ogr.wkbPoint)
```

Now that the layer exists, we need to create any attribute fields that should appear on the layer. Fields must be added to the layer before any features are written. To create a field we initialize an *OGRField* object with the information about the field. In the case of Shapefiles, the field width and precision is significant in the creation of the output .dbf file, so we set it specifically, though generally the defaults are OK. For this example we will just have one attribute, a name string associated with the x,y point.

Note that the template OGRField we pass to *OGRLayer::CreateField()* is copied internally. We retain ownership of the object.

C++

C

Python

```
OGRFieldDefn oField( "Name", OFTString );

oField.SetWidth(32);

if( poLayer->CreateField( &oField ) != OGRERR_NONE )
{
    printf( "Creating Name field failed.\n" );
    exit( 1 );
}
```

```
OGRFieldDefnH hFieldDefn;

hFieldDefn = OGR_Fld_Create( "Name", OFTString );

OGR_Fld_SetWidth( hFieldDefn, 32);

if( OGR_L_CreateField( hLayer, hFieldDefn, TRUE ) != OGRERR_NONE )
{
    printf( "Creating Name field failed.\n" );
    exit( 1 );
}

OGR_Fld_Destroy(hFieldDefn);
```

```
field_defn = ogr.FieldDefn("Name", ogr.OFTString)
field_defn.SetWidth(32)

lyr.CreateField(field_defn)
```

The following snipping loops reading lines of the form "x,y,name" from stdin, and parsing them.

C++

C++

Python

```
double x, y;
char szName[33];

while( !feof(stdin)
       && fscanf( stdin, "%lf,%lf,%32s", &x, &y, szName ) == 3 )
{
```

```
double x, y;
char szName[33];

while( !feof(stdin)
       && fscanf( stdin, "%lf,%lf,%32s", &x, &y, szName ) == 3 )
{
```

```
# Expected format of user input: x y name
linestring = input()
linelist = linestring.split()

while len(linelist) == 3:
  ...
```

To write a feature to disk, we must create a local OGRFeature, set attributes and attach geometry before trying to write it to the layer. It is imperative that this feature be instantiated from the OGRFeatureDefn associated with the layer it will be written to.

C++

C

Python

```
OGRFeature *poFeature;

poFeature = OGRFeature::CreateFeature( poLayer->GetLayerDefn() );
poFeature->SetField( "Name", szName );
```

```
OGRFeatureH hFeature;

hFeature = OGR_F_Create( OGR_L_GetLayerDefn( hLayer ) );
OGR_F_SetFieldString( hFeature, OGR_F_GetFieldIndex(hFeature, "Name"), szName );
```

```
feat = ogr.Feature(lyr.GetLayerDefn())
feat.SetField("Name", name)
```

We create a local geometry object, and assign its copy (indirectly) to the feature. The *OGRFeature::SetGeometryDirectly()* differs from *OGRFeature::SetGeometry()* in that the direct method gives ownership of the geometry to the feature. This is generally more efficient as it avoids an extra deep object copy of the geometry.

C++

C

Python

```
OGRPoint pt;
pt.setX( x );
pt.setY( y );

poFeature->SetGeometry( &pt );
```

```
OGRGeometryH hPt;
hPt = OGR_G_CreateGeometry(wkbPoint);
OGR_G_SetPoint_2D(hPt, 0, x, y);

OGR_F_SetGeometry( hFeature, hPt );
OGR_G_DestroyGeometry(hPt);
```

```
pt = ogr.Geometry(ogr.wkbPoint)
pt.SetPoint_2D(0, x, y)

feat.SetGeometry(pt)
```

Now we create a feature in the file. The *OGRLayer::CreateFeature()* does not take ownership of our feature so we clean it up when done with it.

C++

C

Python

```
     if( poLayer->CreateFeature( poFeature ) != OGRERR_NONE )
     {
         printf( "Failed to create feature in shapefile.\n" );
        exit( 1 );
     }

     OGRFeature::DestroyFeature( poFeature );
}
```

```
     if( OGR_L_CreateFeature( hLayer, hFeature ) != OGRERR_NONE )
     {
         printf( "Failed to create feature in shapefile.\n" );
        exit( 1 );
     }

     OGR_F_Destroy( hFeature );
}
```

```
lyr.CreateFeature(feat)
```

Finally we need to close down the datasource in order to ensure headers are written out in an orderly way and all resources are recovered.

C++

C

Python

```
  GDALClose( poDS );
}
```

```
  GDALClose( poDS );
}
```

```
ds.Close()
```

The same program all in one block looks like this:

C++

C

Python

```
#include "ogrsf_frmts.h"

int main()
{
    const char *pszDriverName = "ESRI Shapefile";
    GDALDriver *poDriver;

    GDALAllRegister();

    poDriver = GetGDALDriverManager()->GetDriverByName(pszDriverName );
    if( poDriver == NULL )
    {
        printf( "%s driver not available.\n", pszDriverName );
        exit( 1 );
    }

    GDALDataset *poDS;

    poDS = poDriver->Create( "point_out.shp", 0, 0, 0, GDT_Unknown, NULL );
    if( poDS == NULL )
    {
        printf( "Creation of output file failed.\n" );
        exit( 1 );
    }

    OGRLayer *poLayer;

    poLayer = poDS->CreateLayer( "point_out", NULL, wkbPoint, NULL );
    if( poLayer == NULL )
    {
        printf( "Layer creation failed.\n" );
        exit( 1 );
    }

    OGRFieldDefn oField( "Name", OFTString );

    oField.SetWidth(32);
```

(continues on next page)

(continued from previous page)

```
    if( poLayer->CreateField( &oField ) != OGRERR_NONE )
    {
        printf( "Creating Name field failed.\n" );
        exit( 1 );
    }

    double x, y;
    char szName[33];

    while( !feof(stdin)
        && fscanf( stdin, "%lf,%lf,%32s", &x, &y, szName ) == 3 )
    {
        OGRFeature *poFeature;

       poFeature = OGRFeature::CreateFeature( poLayer->GetLayerDefn() );
       poFeature->SetField( "Name", szName );

        OGRPoint pt;

        pt.setX( x );
        pt.setY( y );

        poFeature->SetGeometry( &pt );

        if( poLayer->CreateFeature( poFeature ) != OGRERR_NONE )
        {
            printf( "Failed to create feature in shapefile.\n" );
            exit( 1 );
        }

        OGRFeature::DestroyFeature( poFeature );
    }

    GDALClose( poDS );
}
```

```
#include "gdal.h"

int main()
{
    const char *pszDriverName = "ESRI Shapefile";
    GDALDriverH hDriver;
    GDALDatasetH hDS;
    OGRLayerH hLayer;
    OGRFieldDefnH hFieldDefn;
    double x, y;
    char szName[33];

    GDALAllRegister();

    hDriver = GDALGetDriverByName( pszDriverName );
    if( hDriver == NULL )
```

(continues on next page)

(continued from previous page)

```
    {
        printf( "%s driver not available.\n", pszDriverName );
        exit( 1 );
    }

    hDS = GDALCreate( hDriver, "point_out.shp", 0, 0, 0, GDT_Unknown, NULL );
    if( hDS == NULL )
    {
        printf( "Creation of output file failed.\n" );
        exit( 1 );
    }

    hLayer = GDALDatasetCreateLayer( hDS, "point_out", NULL, wkbPoint, NULL );
    if( hLayer == NULL )
    {
        printf( "Layer creation failed.\n" );
        exit( 1 );
    }

    hFieldDefn = OGR_Fld_Create( "Name", OFTString );

    OGR_Fld_SetWidth( hFieldDefn, 32);

    if( OGR_L_CreateField( hLayer, hFieldDefn, TRUE ) != OGRERR_NONE )
    {
        printf( "Creating Name field failed.\n" );
        exit( 1 );
    }

    OGR_Fld_Destroy(hFieldDefn);

    while( !feof(stdin)
        && fscanf( stdin, "%lf,%lf,%32s", &x, &y, szName ) == 3 )
    {
        OGRFeatureH hFeature;
        OGRGeometryH hPt;

        hFeature = OGR_F_Create( OGR_L_GetLayerDefn( hLayer ) );
        OGR_F_SetFieldString( hFeature, OGR_F_GetFieldIndex(hFeature, "Name"), szName );

        hPt = OGR_G_CreateGeometry(wkbPoint);
        OGR_G_SetPoint_2D(hPt, 0, x, y);

        OGR_F_SetGeometry( hFeature, hPt );
        OGR_G_DestroyGeometry(hPt);

        if( OGR_L_CreateFeature( hLayer, hFeature ) != OGRERR_NONE )
        {
        printf( "Failed to create feature in shapefile.\n" );
        exit( 1 );
        }
```

(continues on next page)

(continued from previous page)

```
        OGR_F_Destroy( hFeature );
    }

    GDALClose( hDS );
}
```

```
import sys

from osgeo import gdal, ogr

gdal.UseExceptions()
driverName = "ESRI Shapefile"
drv = gdal.GetDriverByName(driverName)
if drv is None:
    print("%s driver not available." % driverName)
    sys.exit(1)

ds = drv.Create("point_out.shp", 0, 0, 0, gdal.GDT_Unknown)

lyr = ds.CreateLayer("point_out", None, ogr.wkbPoint)

field_defn = ogr.FieldDefn("Name", ogr.OFTString)
field_defn.SetWidth(32)

lyr.CreateField(field_defn)

# Expected format of user input: x y name
linestring = input()
linelist = linestring.split()

while len(linelist) == 3:
    x = float(linelist[0])
    y = float(linelist[1])
    name = linelist[2]

    feat = ogr.Feature(lyr.GetLayerDefn())
    feat.SetField("Name", name)

    pt = ogr.Geometry(ogr.wkbPoint)
    pt.SetPoint_2D(0, x, y)

    feat.SetGeometry(pt)

    lyr.CreateFeature(feat)

    linestring = input()
    linelist = linestring.split()

ds.Close()
```

Several geometry fields can be associated to a feature. This capability is just available for a few file formats, such as PostGIS.

To create such datasources, geometry fields must be first created. Spatial reference system objects can be associated to each geometry field.

C++

C

```
OGRGeomFieldDefn oPointField( "PointField", wkbPoint );
OGRSpatialReference* poSRS = new OGRSpatialReference();
poSRS->importFromEPSG(4326);
oPointField.SetSpatialRef(poSRS);
poSRS->Release();

if( poLayer->CreateGeomField( &oPointField ) != OGRERR_NONE )
{
    printf( "Creating field PointField failed.\n" );
    exit( 1 );
}

OGRGeomFieldDefn oFieldPoint2( "PointField2", wkbPoint );
poSRS = new OGRSpatialReference();
poSRS->importFromEPSG(32631);
oPointField2.SetSpatialRef(poSRS);
poSRS->Release();

if( poLayer->CreateGeomField( &oPointField2 ) != OGRERR_NONE )
{
    printf( "Creating field PointField2 failed.\n" );
    exit( 1 );
}
```

```
OGRGeomFieldDefnH hPointField;
OGRGeomFieldDefnH hPointField2;
OGRSpatialReferenceH hSRS;

hPointField = OGR_GFld_Create( "PointField", wkbPoint );
hSRS = OSRNewSpatialReference( NULL );
OSRImportFromEPSG(hSRS, 4326);
OGR_GFld_SetSpatialRef(hPointField, hSRS);
OSRRelease(hSRS);

if( OGR_L_CreateGeomField( hLayer, hPointField ) != OGRERR_NONE )
{
    printf( "Creating field PointField failed.\n" );
    exit( 1 );
}

OGR_GFld_Destroy( hPointField );

hPointField2 = OGR_GFld_Create( "PointField2", wkbPoint );
OSRImportFromEPSG(hSRS, 32631);
OGR_GFld_SetSpatialRef(hPointField2, hSRS);
OSRRelease(hSRS);
```

(continues on next page)

(continued from previous page)

```
if( OGR_L_CreateGeomField( hLayer, hPointField2 ) != OGRERR_NONE )
{
    printf( "Creating field PointField2 failed.\n" );
    exit( 1 );
}

OGR_GFld_Destroy( hPointField2 );
```

To write a feature to disk, we must create a local OGRFeature, set attributes and attach geometries before trying to write it to the layer. It is imperative that this feature be instantiated from the OGRFeatureDefn associated with the layer it will be written to.

C++

C

Python

```
OGRFeature *poFeature;
OGRGeometry *poGeometry;
char* pszWKT;

poFeature = OGRFeature::CreateFeature( poLayer->GetLayerDefn() );

pszWKT = (char*) "POINT (2 49)";
OGRGeometryFactory::createFromWkt( &pszWKT, NULL, &poGeometry );
poFeature->SetGeomFieldDirectly( "PointField", poGeometry );

pszWKT = (char*) "POINT (500000 4500000)";
OGRGeometryFactory::createFromWkt( &pszWKT, NULL, &poGeometry );
poFeature->SetGeomFieldDirectly( "PointField2", poGeometry );

if( poLayer->CreateFeature( poFeature ) != OGRERR_NONE )
{
    printf( "Failed to create feature.\n" );
    exit( 1 );
}

OGRFeature::DestroyFeature( poFeature );
```

```
OGRFeatureH hFeature;
OGRGeometryH hGeometry;
char* pszWKT;

poFeature = OGR_F_Create( OGR_L_GetLayerDefn(hLayer) );

pszWKT = (char*) "POINT (2 49)";
OGR_G_CreateFromWkt( &pszWKT, NULL, &hGeometry );
OGR_F_SetGeomFieldDirectly( hFeature,
    OGR_F_GetGeomFieldIndex(hFeature, "PointField"), hGeometry );

pszWKT = (char*) "POINT (500000 4500000)";
OGR_G_CreateFromWkt( &pszWKT, NULL, &hGeometry );
```

(continues on next page)

(continued from previous page)

```
OGR_F_SetGeomFieldDirectly( hFeature,
    OGR_F_GetGeomFieldIndex(hFeature, "PointField2"), hGeometry );

if( OGR_L_CreateFeature( hFeature ) != OGRERR_NONE )
{
    printf( "Failed to create feature.\n" );
    exit( 1 );
}

OGR_F_Destroy( hFeature );
```

```
feat = ogr.Feature( lyr.GetLayerDefn() )

feat.SetGeomFieldDirectly( "PointField",
    ogr.CreateGeometryFromWkt( "POINT (2 49)" ) )
feat.SetGeomFieldDirectly( "PointField2",
    ogr.CreateGeometryFromWkt( "POINT (500000 4500000)" ) )

if lyr.CreateFeature( feat ) != 0:
    print( "Failed to create feature.\n" );
    sys.exit( 1 );
```

10.3.1.4 Writing to OGR using the Arrow C Data interface

Added in version 3.8.

Instead of writing features one at a time, it is also possible to write them by batches, with a column-oriented memory layout, using the *OGRLayer::WriteArrowBatch()* method. Note that this method is more difficult to use than the traditional *OGRLayer::CreateFeature()* approach, and is only advised when compatibility with the Apache Arrow C Data interface is needed, or when column-oriented writing of layers is required.

Pending using an helper library, generation of the Arrow C Data interface requires reading of the following documents:

- Arrow C data interface
- Arrow Columnar Format.

Consult *Reading From OGR using the Arrow C Stream data interface* for introduction to the ArrowSchema and ArrowArray basic types involved for batch writing.

The WriteArrowBatch() method has the following signature:

```
/** Writes a batch of rows from an ArrowArray.
 *
 * @param schema Schema of array
 * @param array Array of type struct. It may be released (array->release==NULL)
 *              after calling this method.
 * @param papszOptions Options. Null terminated list, or nullptr.
 * @return true in case of success
 */
virtual bool OGRLayer::WriteArrowBatch(const struct ArrowSchema *schema,
                                       struct ArrowArray *array,
                                       CSLConstList papszOptions = nullptr);
```

It is also available in the C API as *OGR_L_WriteArrowBatch()*.

This is semantically close to calling *OGRLayer::CreateFeature()* with multiple features at once.

The ArrowArray must be of type struct (format=+s), and its children generally map to a OGR attribute or geometry field (unless they are struct themselves).

Method *OGRLayer::IsArrowSchemaSupported()* can be called to determine if the schema will be supported by WriteArrowBatch().

OGR fields for the corresponding children arrays must exist and be of a compatible type. For attribute fields, they should be created with *OGRLayer::CreateFieldFromArrowSchema()*.

Arrays for geometry columns should be of binary or large binary type and contain WKB geometry.

Note that the passed array may be set to a released state (array->release==NULL) after this call (not by the base implementation, but in specialized ones such as Parquet or Arrow for example)

Supported options of the base implementation are:

- FID=name. Name of the FID column in the array. If not provided, GetFIDColumn() is used to determine it. The special name OGRLayer::DEFAULT_ARROW_FID_NAME is also recognized if neither FID nor GetFIDColumn() are set. The corresponding ArrowArray must be of type int32 (i) or int64 (l). On input, values of the FID column are used to create the feature. On output, the values of the FID column may be set with the FID of the created feature (if the array is not released).
- GEOMETRY_NAME=name. Name of the geometry column. If not provided, GetGeometryColumn() is used. The special name OGRLayer::DEFAULT_ARROW_GEOMETRY_NAME is also recognized if neither GEOMETRY_NAME nor GetGeometryColumn() are set. Geometry columns are also identified if they have ARROW:extension:name=ogc.wkb as a field metadata. The corresponding ArrowArray must be of type binary (w) or large binary (W).

Drivers that have a specialized implementation (such as *(Geo)Parquet* and *(Geo)Arrow IPC File Format / Stream*) advertise the OLCFastWriteArrowBatch layer capability.

The following example in Python demonstrates how to copy a layer from one format to another one (assuming it has at most a single geometry column):

```
def copy_layer(src_lyr, out_filename, out_format, lcos = {}):
    stream = src_lyr.GetArrowStream()
    schema = stream.GetSchema()

    # If the source layer has a FID column and the output driver supports
    # a FID layer creation option, set it to the source FID column name.
    if src_lyr.GetFIDColumn():
        creationOptions = gdal.GetDriverByName(out_format).GetMetadataItem(
            "DS_LAYER_CREATIONOPTIONLIST"
        )
        if creationOptions and '"FID"' in creationOptions:
            lcos["FID"] = src_lyr.GetFIDColumn()

    with ogr.GetDriverByName(out_format).CreateDataSource(out_filename) as out_ds:
        if src_lyr.GetLayerDefn().GetGeomFieldCount() > 1:
            out_lyr = out_ds.CreateLayer(
                src_lyr.GetName(), geom_type=ogr.wkbNone, options=lcos
            )
            for i in range(src_lyr.GetLayerDefn().GetGeomFieldCount()):
                out_lyr.CreateGeomField(src_lyr.GetLayerDefn().GetGeomFieldDefn(i))
        else:
            out_lyr = out_ds.CreateLayer(
```

(continues on next page)

(continued from previous page)

```
                src_lyr.GetName(),
                geom_type=src_lyr.GetGeomType(),
                srs=src_lyr.GetSpatialRef(),
                options=lcos,
            )

        success, error_msg = out_lyr.IsArrowSchemaSupported(schema)
        assert success, error_msg

        src_geom_field_names = [
            src_lyr.GetLayerDefn().GetGeomFieldDefn(i).GetName()
            for i in range(src_lyr.GetLayerDefn().GetGeomFieldCount())
        ]
        for i in range(schema.GetChildrenCount()):
            # GetArrowStream() may return "OGC_FID" for a unnamed source FID
            # column and "wkb_geometry" for a unnamed source geometry column.
            # Also test GetFIDColumn() and src_geom_field_names if they are
            # named.
            if (
                schema.GetChild(i).GetName()
                not in ("OGC_FID", "wkb_geometry", src_lyr.GetFIDColumn())
                and schema.GetChild(i).GetName() not in src_geom_field_names
            ):
                out_lyr.CreateFieldFromArrowSchema(schema.GetChild(i))

        write_options = []
        if src_lyr.GetFIDColumn():
            write_options.append("FID=" + src_lyr.GetFIDColumn())
        if (
            src_lyr.GetLayerDefn().GetGeomFieldCount() == 1
            and src_lyr.GetGeometryColumn()
        ):
            write_options.append("GEOMETRY_NAME=" + src_lyr.GetGeometryColumn())

        while True:
            array = stream.GetNextRecordBatch()
            if array is None:
                break
            out_lyr.WriteArrowBatch(schema, array, write_options)
```

For the Python bindings, in addition to the above ogr.Layer.IsArrowSchemaSupported(), ogr.Layer.CreateFieldFromArrowSchema() and ogr.Layer.WriteArrowBatch() methods, 3 similar methods exist using the PyArrow data types:

```
class Layer:

    def IsPyArrowSchemaSupported(self, pa_schema, options=[]):
        """Returns whether the passed pyarrow Schema is supported by the layer, as a␣
↪tuple (success: bool, errorMsg: str).

    def CreateFieldFromPyArrowSchema(self, pa_schema, options=[]):
        """Create a field from the passed pyarrow Schema."""
```

(continues on next page)

(continued from previous page)

```
    def WritePyArrow(self, pa_batch, options=[]):
        """Write the content of the passed PyArrow batch (either a pyarrow.Table, a␣
↪pyarrow.RecordBatch or a pyarrow.StructArray) into the layer."""
```

10.3.2 Vector driver implementation tutorial

10.3.2.1 Overall Approach

In general new formats are added to OGR by implementing format specific drivers with instantiating a *GDALDriver* and subclasses of *GDALDataset* and *OGRLayer*. The GDALDriver instance is registered with the *GDALDriverManager* at runtime.

Before following this tutorial to implement an OGR driver, please review the *Vector Data Model* document carefully.

The tutorial will be based on implementing a simple ascii point format.

10.3.2.2 Implementing GDALDriver

The format specific driver class is implemented as a instance of GDALDriver. One instance of the driver will normally be created, and registered with the GDALDriverManager. The instantiation of the driver is normally handled by a global C callable registration function, similar to the following placed in the same file as the driver class.

```
void RegisterOGRSPF()
{
    if( GDALGetDriverByName("SPF") != NULL )
        return;

    GDALDriver *poDriver = new GDALDriver();

    poDriver->SetDescription("SPF");
    poDriver->SetMetadataItem(GDAL_DCAP_VECTOR, "YES");
    poDriver->SetMetadataItem(GDAL_DMD_LONGNAME, "Long name for SPF driver");
    poDriver->SetMetadataItem(GDAL_DMD_EXTENSION, "spf");
    poDriver->SetMetadataItem(GDAL_DMD_HELPTOPIC, "drv_spf.html");

    poDriver->pfnOpen = OGRSPFDriverOpen;
    poDriver->pfnIdentify = OGRSPFDriverIdentify;
    poDriver->pfnCreate = OGRSPFDriverCreate;

    poDriver->SetMetadataItem(GDAL_DCAP_VIRTUALIO, "YES");

    GetGDALDriverManager()->RegisterDriver(poDriver);
}
```

The `GDALDriver::SetDescription()` sets the name of the driver. This name is specified on the commandline when creating datasources so it is generally good to keep it short and without any special characters or spaces.

SetMetadataItem(GDAL_DCAP_VECTOR, "YES") is specified to indicate that the driver will handle vector data.

SetMetadataItem(GDAL_DCAP_VIRTUALIO, "YES") is specified to indicate that the driver can deal with files opened with the VSI*L GDAL API. Otherwise this metadata item should not be defined.

For a driver that can be built as a plugin (that is to say a standalone shared object, loaded at runtime by GDAL), since GDAL 3.9 and rfc-96, there is a way to implement the driver in a way where the plugin will be loaded only when

necessary, and not immediately at *GDALAllRegister()* time. Consult rfc96_example_driver for the changes needed to make the driver compatible with deferred plugin loading.

The driver declaration generally looks something like this for a format with read or read and update access (the Open() method) and creation support (the Create() method).

```
static GDALDataset* OGRSPFDriverOpen(GDALOpenInfo* poOpenInfo);
static int          OGRSPFDriverIdentify(GDALOpenInfo* poOpenInfo);
static GDALDataset* OGRSPFDriverCreate(const char* pszName, int nXSize, int nYSize,
                                       int nBands, GDALDataType eDT, char** papszOptions);
```

The Open() method is called by *GDALOpenEx()*. It should quietly return NULL if the passed filename is not of the format supported by the driver. If it is the target format, then a new GDALDataset object for the dataset should be returned.

It is common for the Open() method to be delegated to an Open() method on the actual format's GDALDataset class.

```
static GDALDataset *OGRSPFDriverOpen( GDALOpenInfo* poOpenInfo )
{
    if( !OGRSPFDriverIdentify(poOpenInfo) )
        return NULL;

    OGRSPFDataSource *poDS = new OGRSPFDataSource();
    if( !poDS->Open(poOpenInfo->pszFilename, poOpenInfo->eAccess == GA_Update) )
    {
        delete poDS;
        return NULL;
    }

    return poDS;
}
```

The Identify() method is implemented as such :

```
static int OGRSPFDriverIdentify( GDALOpenInfo* poOpenInfo )
{
    // Does this appear to be an .spf file?
    return EQUAL(CPLGetExtension(poOpenInfo->pszFilename), "spf");
}
```

Examples of the Create() method is left for the section on creation and update.

10.3.2.3 Basic Read Only Data Source

We will start implementing a minimal read-only datasource. No attempt is made to optimize operations, and default implementations of many methods inherited from GDALDataset are used.

The primary responsibility of the datasource is to manage the list of layers. In the case of the SPF format a datasource is a single file representing one layer so there is at most one layer. The "name" of a datasource should generally be the name passed to the Open() method.

The Open() method below is not overriding a base class method, but we have it to implement the open operation delegated by the driver class.

For this simple case we provide a stub *GDALDataset::TestCapability()* that returns FALSE for all extended capabilities. The TestCapability() method is pure virtual, so it does need to be implemented.

```
class OGRSPFDataSource : public GDALDataset
{
    OGRSPFLayer       **papoLayers;
    int                 nLayers;

public:
                        OGRSPFDataSource();
                        ~OGRSPFDataSource();

    int                 Open( const char *pszFilename, int bUpdate );

    int                 GetLayerCount() { return nLayers; }
    OGRLayer            *GetLayer( int );

    int                 TestCapability( const char * ) { return FALSE; }
};
```

The constructor is a simple initializer to a default state. The Open() will take care of actually attaching it to a file. The destructor is responsible for orderly cleanup of layers.

```
OGRSPFDataSource::OGRSPFDataSource()
{
    papoLayers = NULL;
    nLayers = 0;
}

OGRSPFDataSource::~OGRSPFDataSource()
{
    for( int i = 0; i < nLayers; i++ )
        delete papoLayers[i];
    CPLFree(papoLayers);
}
```

The Open() method is the most important one on the datasource, though in this particular instance it passes most of its work off to the OGRSPFLayer constructor if it believes the file is of the desired format.

Note that Open() methods should try and determine that a file isn't of the identified format as efficiently as possible, since many drivers may be invoked with files of the wrong format before the correct driver is reached. In this particular Open() we just test the file extension but this is generally a poor way of identifying a file format. If available, checking "magic header values" or something similar is preferable.

In the case of the SPF format, update in place is not supported, so we always fail if bUpdate is FALSE.

```
int  OGRSPFDataSource::Open( const char *pszFilename, int bUpdate )
{
    if( bUpdate )
    {
        CPLError(CE_Failure, CPLE_OpenFailed,
                "Update access not supported by the SPF driver.");
        return FALSE;
    }

    // Create a corresponding layer.
    nLayers = 1;
```

(continues on next page)

(continued from previous page)

```
    papoLayers = static_cast<OGRSPFLayer **>(CPLMalloc(sizeof(void *)));

    papoLayers[0] = new OGRSPFLayer(pszFilename);

    pszName = CPLStrdup(pszFilename);

    return TRUE;
}
```

A GetLayer() method also needs to be implemented. Since the layer list is created in the Open() this is just a lookup with some safety testing.

```
OGRLayer *OGRSPFDataSource::GetLayer( int iLayer )
{
    if( iLayer < 0 || iLayer >= nLayers )
        return NULL;

    return papoLayers[iLayer];
}
```

10.3.2.4 Read Only Layer

The OGRSPFLayer is implements layer semantics for an .spf file. It provides access to a set of feature objects in a consistent coordinate system with a particular set of attribute columns. Our class definition looks like this:

```
class OGRSPFLayer : public OGRLayer
{
    OGRFeatureDefn     *poFeatureDefn;
    FILE               *fp;
    int                 nNextFID;

public:
    OGRSPFLayer( const char *pszFilename );
~OGRSPFLayer();

    void                ResetReading();
    OGRFeature *        GetNextFeature();

    OGRFeatureDefn *    GetLayerDefn() { return poFeatureDefn; }

    int                 TestCapability( const char * ) { return FALSE; }
};
```

The layer constructor is responsible for initialization. The most important initialization is setting up the *OGRFeatureDefn* for the layer. This defines the list of fields and their types, the geometry type and the coordinate system for the layer. In the SPF format the set of fields is fixed - a single string field and we have no coordinate system info to set.

Pay particular attention to the reference counting of the OGRFeatureDefn. As OGRFeature's for this layer will also take a reference to this definition, it is important that we also establish a reference on behalf of the layer itself.

```
OGRSPFLayer::OGRSPFLayer( const char *pszFilename )
{
```

(continues on next page)

(continued from previous page)

```
    nNextFID = 0;

    poFeatureDefn = new OGRFeatureDefn(CPLGetBasename(pszFilename));
    SetDescription(poFeatureDefn->GetName());
    poFeatureDefn->Reference();
    poFeatureDefn->SetGeomType(wkbPoint);

    OGRFieldDefn oFieldTemplate("Name", OFTString);

    poFeatureDefn->AddFieldDefn(&oFieldTemplate);

    fp = VSIFOpenL(pszFilename, "r");
    if( fp == NULL )
        return;
}
```

Note that the destructor uses *OGRFeatureDefn::Release()* on the OGRFeatureDefn. This will destroy the feature definition if the reference count drops to zero, but if the application is still holding onto a feature from this layer, then that feature will hold a reference to the feature definition and it will not be destroyed here (which is good!).

```
OGRSPFLayer::~OGRSPFLayer()
{
    poFeatureDefn->Release();
    if( fp != NULL )
        VSIFCloseL(fp);
}
```

The *OGRLayer::GetNextFeature()* method is usually the work horse of OGRLayer implementations. It is responsible for reading the next feature according to the current spatial and attribute filters installed.

The while() loop is present to loop until we find a satisfactory feature. The first section of code is for parsing a single line of the SPF text file and establishing the x, y and name for the line.

```
OGRFeature *OGRSPFLayer::GetNextFeature()
{
    // Loop till we find a feature matching our requirements.
    while( true )
    {
        const char *pszLine = CPLReadLineL(fp);

        // Are we at end of file (out of features)?
        if( pszLine == NULL )
            return NULL;

        const double dfX = atof(pszLine);

        pszLine = strstr(pszLine,"|");
        if( pszLine == NULL )
            continue; // we should issue an error!
        else
            pszLine++;

        const double dfY = atof(pszLine);
```

(continues on next page)

(continued from previous page)

```
        pszLine = strstr(pszLine,"|");

        const char *pszName = NULL;
        if( pszLine == NULL )
            continue; // we should issue an error!
        else
            pszName = pszLine + 1;
```

The next section turns the x, y and name into a feature. Also note that we assign a linearly incremented feature id. In our case we started at zero for the first feature, though some drivers start at 1.

```
OGRFeature *poFeature = new OGRFeature(poFeatureDefn);

poFeature->SetGeometryDirectly(new OGRPoint(dfX, dfY));
poFeature->SetField(0, pszName);
poFeature->SetFID(nNextFID++);
```

Next we check if the feature matches our current attribute or spatial filter if we have them. Methods on the OGRLayer base class support maintain filters in the OGRLayer member fields `OGRLayer::m_poFilterGeom` (spatial filter) and `OGRLayer::m_poAttrQuery` (attribute filter) so we can just use these values here if they are non-NULL. The following test is essentially "stock" and done the same in all formats. Some formats also do some spatial filtering ahead of time using a spatial index.

If the feature meets our criteria we return it. Otherwise we destroy it, and return to the top of the loop to fetch another to try.

```
        if( (m_poFilterGeom == NULL ||
            FilterGeometry(poFeature->GetGeometryRef())) &&
            (m_poAttrQuery == NULL ||
            m_poAttrQuery->Evaluate(poFeature)) )
            return poFeature;

        delete poFeature;
    }
}
```

While in the middle of reading a feature set from a layer, or at any other time the application can call *`OGRLayer::ResetReading()`* which is intended to restart reading at the beginning of the feature set. We implement this by seeking back to the beginning of the file, and resetting our feature id counter.

```
void OGRSPFLayer::ResetReading()
{
    VSIFSeekL(fp, 0, SEEK_SET);
    nNextFID = 0;
}
```

In this implementation we do not provide a custom implementation for the GetFeature() method. This means an attempt to read a particular feature by its feature id will result in many calls to GetNextFeature() until the desired feature is found. However, in a sequential text format like spf there is little else we could do anyway.

There! We have completed a simple read-only feature file format driver.

10.3.3 Downloading GeoParquet Data from Overture Maps

Last Updated
2025-09-26

Version Tested
GDAL 3.11.4

Contents

- *Downloading GeoParquet Data from Overture Maps*
  - *Overview*
  - *Setting Up a Conda Environment with Parquet Support*
  - *Listing Overture Data*
  - *Downloading Data*

10.3.3.1 Overview

This tutorial guides you through a complete workflow for downloading data from Overture Maps using GDAL's *(Geo)Parquet* driver. We'll use the GDAL command-line tools installed via Conda, to download building and administrative division data for Mostar, Bosnia and Herzegovina, the host city for FOSS4G Europe 2025.

> **Note**
> As Overture only maintain publicly available datasets for a maximum of 60 days (two monthly releases) the data version URLs may need to be updated (by changing the release name `2025-09-24.0`). Use the `gdal vsi list` command detailed below to get the latest available versions. See the Overture Data retention policy for details.

Code examples assume a Bash shell on Linux. Where syntax differs, PowerShell examples are also provided for Windows users. The tutorial covers both the unified *gdal command-line interface* and the *traditional* individual GDAL tools.

10.3.3.2 Setting Up a Conda Environment with Parquet Support

This tutorial assumes Conda is already installed. We'll start by creating a new Conda environment and installing GDAL.

```
$ conda create --yes --name gdal
$ conda activate gdal
$ conda install --yes -c conda-forge gdal
$ conda --version
conda 24.3.0
$ gdal --version
GDAL 3.11.4 "Eganville", released 2025/09/04
```

> **Note**
> Be sure to install GDAL from the `conda-forge` channel, as it typically provides more up-to-date versions than the `defaults` channel. For example, at the time of writing the `defaults` channel offers GDAL 3.6.2, while `conda-forge` provides version 3.11.4.

Now that GDAL is installed, let's verify that it supports reading and writing Parquet files.

gdal CLI

Traditional

```
gdal vector --formats | grep Parquet

# in PowerShell we can use
# gdal vector --formats | Select-String "Parquet"

# or in a JSON format
gdal vector --drivers
```

```
ogrinfo --formats
# search in the output
ogrinfo --formats | grep Parquet
```

If you don't see a Parquet driver listed in the output above, don't worry - it's an optional plugin. As noted on the *GeoParquet* driver page, support can be added separately via a package available on `conda-forge`. Once installed, the driver should appear in the list of supported formats.

```
$ conda install --yes -c conda-forge libgdal-arrow-parquet
$ gdal vector --formats | grep Parquet
Parquet -vector- (rw+v): (Geo)Parquet (*.parquet)
```

Note

On Windows, because GDAL was installed after the environment was activated, some required environment variables may not be set. If you try to use the Parquet driver now, you might encounter an error like the one below:

```
ERROR 4: `C:\Users\GDAL\AppData\Local\Temp\_gdal_http_example.parquet'
not recognized as being in a supported file format. It could have been recognized by␣
↪driver Parquet, but plugin ogr_Parquet.dll is not available in your installation.
You may install it with 'conda install -c conda-forge libgdal-arrow-parquet'. The␣
↪GDAL_DRIVER_PATH configuration option is not set.
```

To fix this, simply reactivate the Conda environment. Going forward, these environment variables will be set automatically each time the environment is activated.

```
conda activate gdal
# check the library is installed
conda list | Select-String "libgdal-arrow-parquet"
# libgdal-arrow-parquet     3.11.4               h33343fa_2    conda-forge
```

Now that the driver is installed, let's verify it by retrieving information from a sample Parquet dataset in the GDAL repository.

gdal CLI

Traditional

```
$ gdal vector info https://github.com/OSGeo/gdal/raw/refs/heads/master/autotest/ogr/data/
↪parquet/example.parquet -f json
{
```

(continues on next page)

(continued from previous page)

```
  "description":"https://github.com/OSGeo/gdal/raw/refs/heads/master/autotest/ogr/data/
↪parquet/example.parquet",
  "driverShortName":"Parquet",
  "driverLongName":"(Geo)Parquet",
  "layers":[
    {
      "name":"_gdal_http_example",
      ...
```

```
$ ogrinfo https://github.com/OSGeo/gdal/raw/refs/heads/master/autotest/ogr/data/parquet/
↪example.parquet
INFO: Open of `https://github.com/OSGeo/gdal/raw/refs/heads/master/autotest/ogr/data/
↪parquet/example.parquet'
      using driver `Parquet' successful.
1: _gdal_http_example (Multi Polygon)
```

10.3.3.3 Listing Overture Data

Overture data is hosted in public Amazon S3 buckets. GDAL can be used to access and browse the contents of these buckets. Although the data is publicly accessible and doesn't require authentication, you'll need to set the `AWS_NO_SIGN_REQUEST` configuration option to prevent GDAL from attempting to sign the request.

```
$ gdal vsi list "/vsis3/overturemaps-us-west-2/release/" --config AWS_NO_SIGN_
↪REQUEST=YES -f json
[
  "2025-08-20.0",
  "2025-08-20.1",
  "2025-09-24.0"
]
```

If the configuration option is not provided, an error will occur:

```
$ gdal vsi list "/vsis3/overturemaps-us-west-2/release/"
ERROR 3: list: '/vsis3/overturemaps-us-west-2/release/' cannot be accessed.␣
↪InvalidCredentials: No valid AWS credentials found.
For authenticated requests, you need to set AWS_SECRET_ACCESS_KEY, AWS_ACCESS_KEY_ID or␣
↪other configuration options,
or create a /home/even/.aws/credentials file.
Consult https://gdal.org/en/stable/user/virtual_file_systems.html#vsis3-aws-s3-files for␣
↪more details.
For unauthenticated requests on public resources, set the AWS_NO_SIGN_REQUEST␣
↪configuration option to YES.
```

To avoid including the `--config AWS_NO_SIGN_REQUEST=YES` option in every command, you can set an environment variable instead:

Bash

PowerShell

```
export AWS_NO_SIGN_REQUEST="YES"
gdal vsi list "/vsis3/overturemaps-us-west-2/release/"
```

```
$ENV:AWS_NO_SIGN_REQUEST="YES"
gdal vsi list "/vsis3/overturemaps-us-west-2/release/"
```

The *gdal vsi* commands were introduced in GDAL 3.11. Before that, a helper Python script included with the GDAL installation was used to list the contents of an S3 bucket - and remains available until GDAL 3.12. To run the `gdal_ls.py` script, navigate to the `samples` directory, set the `AWS_NO_SIGN_REQUEST` environment variable, and execute the script using Python:

Bash

PowerShell

```
PYTHON_VERSION=$(python -c 'import sys; print(f"{sys.version_info.major}.{sys.version_
↪info.minor}")')
cd "$CONDA_PREFIX/lib/python$PYTHON_VERSION/site-packages/osgeo_utils/samples"
export AWS_NO_SIGN_REQUEST="YES"
python gdal_ls.py "/vsis3/overturemaps-us-west-2/release/"
```

```
cd "$ENV:CONDA_PREFIX\Lib\site-packages\osgeo_utils\samples"
$ENV:AWS_NO_SIGN_REQUEST="YES"
python gdal_ls.py "/vsis3/overturemaps-us-west-2/release/"
```

You can list objects at different levels of the hierarchy by changing the input path or by using the `--recursive` and `--depth` options available in *gdal vsi list*.

```
$ gdal vsi list "/vsis3/overturemaps-us-west-2/release/2025-09-24.0/theme=buildings/
↪type=building"
[
  "part-00000-0df994ca-3323-4d7c-a374-68c653f78289-c000.zstd.parquet",
  "part-00001-0df994ca-3323-4d7c-a374-68c653f78289-c000.zstd.parquet",
  ...

$ gdal vsi list "/vsis3/overturemaps-us-west-2/release/2025-09-24.0" --recursive --depth␣
↪2
[
  "theme=addresses",
  "theme=addresses/type=address",
  "theme=base",
  "theme=base/type=bathymetry",
  ...
```

Note the versions used above may need to be updated, as Overture only retail each version for 2 months. If a version has been removed you will see an error such as:

```
ERROR 3: list: '/vsis3/overturemaps-us-west-2/release/2025-09-24.0/theme=buildings/
↪type=building' does not exist
```

To view details for any of the Parquet files:

gdal CLI

Traditional

```
$ gdal vector info "/vsis3/overturemaps-us-west-2/release/2025-09-24.0/theme=buildings/
↪type=building/part-00212-93d280c5-9152-4ecc-b500-d9681495941f-c000.zstd.parquet"
```

(continues on next page)

(continued from previous page)

```
{
  "description":"/vsis3/overturemaps-us-west-2/release/2025-09-24.0/theme=buildings/
↪type=building/part-00212-93d280c5-9152-4ecc-b500-d9681495941f-c000.zstd.parquet",
  "driverShortName":"Parquet",
  "driverLongName":"(Geo)Parquet",
  "layers":[
    {
      "name":"part-00212-93d280c5-9152-4ecc-b500-d9681495941f-c000.zstd",
      ...
```

```
$ ogrinfo "/vsis3/overturemaps-us-west-2/release/2025-09-24.0/theme=buildings/
↪type=building/part-00212-93d280c5-9152-4ecc-b500-d9681495941f-c000.zstd.parquet"
INFO: Open of `/vsis3/overturemaps-us-west-2/release/2025-09-24.0/theme=buildings/
↪type=building/part-00212-93d280c5-9152-4ecc-b500-d9681495941f-c000.zstd.parquet'
      using driver `Parquet' successful.
1: part-00212-0df994ca-3323-4d7c-a374-68c653f78289-c000.zstd (Multi Polygon)
```

You can also explore the available data using the map interface at https://explore.overturemaps.org/.

10.3.3.4 Downloading Data

Now that we've explored the available data and its organization, let's download building polygons for the Mostar region. As we saw when listing the data, it's split into hundreds of individual files with names like `part-00212-0df994ca-3323-4d7c-a374-68c653f78289-c000.zstd`. However, since version 3.6, GDAL can read entire directories of Parquet files and treat them as a single layer. To enable this, specify the `PARQUET` driver as part of the input string.

Let's apply a spatial filter using a bounding box to retrieve data. Because of the amount of remote data, this operation may take a few minutes to complete.

gdal CLI

Traditional

```
$ gdal vector filter \
    --input "PARQUET:/vsis3/overturemaps-us-west-2/release/2025-09-24.0/theme=buildings/
↪type=building/" \
    --bbox 17.773,43.331,17.8392,43.3716 \
    --output buildings.parquet
```

```
$ ogr2ogr buildings.parquet \
    "PARQUET:/vsis3/overturemaps-us-west-2/release/2025-09-24.0/theme=buildings/
↪type=building/" \
    -clipsrc 17.773 43.331 17.8392 43.3716
```

> **Note**
>
> When using PowerShell on Windows, use the backtick ` as the line continuation character instead of the backslash \. Other than that, the commands are the same.

Now let's check the output. While you can open the file in a GIS application like QGIS, you can also quickly preview it by "burning" the vector data into an image using GDAL using *rasterize*. We use the `init` option to set a white background, and then pass in RGB values as 3 `burn` parameters.

gdal CLI

Traditional

```
$ gdal vector rasterize \
    --burn 170 --burn 74 --burn 68 \
    --size 400,400 \
    --output-data-type "Byte" \
    --init 255 \
    buildings.parquet buildings.tif

# now convert to a PNG
$ gdal raster convert buildings.tif buildings.png
```

```
$ gdal_rasterize \
    -burn 170 -burn 74 -burn 68 \
    -ot Byte \
    -ts 400 400 \
    -init 255 \
    buildings.parquet buildings.tif

$ gdal_translate -of PNG buildings.tif buildings.png
```

We can also apply attribute filters to control which data is downloaded. In this final example, we'll download administrative divisions for Bosnia and Herzegovina (country code "BA"), limited to the "county" subtype. According to the Divisions documentation, this represents *"the largest sub-region administrative unit in most countries, unless they have macrocounties."*

gdal CLI

Traditional

```
gdal vector filter \
    --input "PARQUET:/vsis3/overturemaps-us-west-2/release/2025-09-24.0/theme=divisions/
↪type=division_area/" \
    --where "country='BA' AND subtype='county'" \
    --output counties.parquet
```

```
ogr2ogr counties.parquet \
    "PARQUET:/vsis3/overturemaps-us-west-2/release/2025-09-24.0/theme=divisions/
↪type=division_area/" \
    -where "country='BA' AND subtype='county'"
```

Finally, let's generate a quick image preview of the data:

```
$ gdal vector rasterize \
    --burn 50 --burn 50 --burn 50 \
    --size 400,400 \
    --output-data-type "Byte" \
    --init 255 \
    counties.parquet counties.tif
$ gdal raster convert counties.tif counties.png
```

10.3.4 Working with DuckDB

Last Updated
2025-09-26

Version Tested
GDAL 3.11.4 - Linux

Version Tested
GDAL 3.12.0dev - Windows

Contents

- *Working with DuckDB*
  - *Overview*
  - *Setup and Configuration*
  - *Downloading Postal Addresses*
    - *Downloading the Data*
    - *Visualizing the Data*
  - *Creating Hex Bins of Point Data*
    - *Downloading the Data*
    - *Visualizing the Data*

10.3.4.1 Overview

This tutorial introduces how to use GDAL together with DuckDB, an open-source, in-process database engine. DuckDB enables querying of remote files directly, so you only download the subset of data you actually need.

In GDAL, DuckDB is accessed via the *ADBC -- Arrow Database Connectivity* driver.

This tutorial provides two examples showing how DuckDB can be used to query, download, and visualize remote datasets from Overture Maps.

10.3.4.2 Setup and Configuration

First, create a new Conda environment and install GDAL along with the required GDAL *Conda subpackage* libgdal-arrow-parquet, which provides Arrow and Parquet support.

On Linux, you will also need to install libgdal-adbc, which is required for the *ADBC -- Arrow Database Connectivity* driver. This package is available via Conda on Linux and is not required on Windows, since DuckDB support is built into the Windows ADBC driver.

Finally, install the DuckDB package. This is also only available via Conda on Linux. For Windows, you will need to install DuckDB manually in a separate step.

Bash

PowerShell

```
conda create --yes --name gdal
conda activate gdal
```

(continues on next page)

(continued from previous page)

```
conda install --yes -c conda-forge gdal libgdal-arrow-parquet libgdal-adbc
conda install --yes libduckdb
```

```
conda create --yes --name gdal
conda activate gdal
# the tutorial only works on Windows with GDAL 3.12 or higher
conda install --yes -c gdal-master -c conda-forge gdal-master::libgdal-arrow-parquet
```

On Windows, DuckDB can be installed by downloading a binary release from the GitHub Releases page.

Be sure to download the shared library `libduckdb.dll` rather than the client package `duckdb_cli-windows-amd64` (which only provides the `duckdb.exe` command-line tool).

After downloading and unzipping the archive, add the folder containing `libduckdb.dll` to your Windows `PATH` so that GDAL can locate it.

```
# download the zip file
$ProgressPreference = "SilentlyContinue"
Invoke-WebRequest `
    -Uri "https://github.com/duckdb/duckdb/releases/download/v1.3.2/libduckdb-windows-
↪amd64.zip" `
    -OutFile "libduckdb-windows-amd64.zip"

# unzip the contents
Expand-Archive -Path "libduckdb-windows-amd64.zip" -DestinationPath "libduckdb-windows-
↪amd64" -Force

# add the unzipped folder to PATH (for this session only)
$absPath = (Resolve-Path ".\libduckdb-windows-amd64").Path
$env:PATH = "$absPath;$env:PATH"
```

Now, let's verify that all required components have been installed correctly in the environment.

Bash

PowerShell

```
gdal --version
# GDAL 3.11.4 "Eganville", released 2025/09/04
conda list | grep "libgdal-arrow-parquet"
# libgdal-arrow-parquet     3.11.4               he2d30bd_3    conda-forge
conda list | grep "duckdb"
# libduckdb                 1.3.2                h8eb67b4_2    conda-forge
gdal vector --formats | grep ADBC
# ADBC -vector- (ro): Arrow Database Connectivity

# test the parquet driver is working
gdal vector info "/vsicurl/https://github.com/opengeospatial/geoparquet/raw/main/
↪examples/example.parquet"
```

```
# the GDAL environment variables GDAL_DRIVER_PATH and GDAL_DATA don't seem to be set␣
↪after installing
# to an active environment, so reactivate it
conda deactivate
```

(continues on next page)

(continued from previous page)

```
conda activate gdal
gdal --version
# GDAL 3.12.0dev-2b1c4d1fdb-dirty, released 2025/09/22

gdal vector --formats | Select-String "ADBC"
# ADBC -vector- (ro): Arrow Database Connectivity

# test the parquet driver is working
gdal vector info "/vsicurl/https://github.com/opengeospatial/geoparquet/raw/main/
↪examples/example.parquet"
```

10.3.4.3 Downloading Postal Addresses

In this first example, we will download addresses from the Overture Addresses Dataset, which contains over 446 million address point entries.

Using DuckDB, we can query and select only the addresses within a specific bounding box in the US, and then download the results to a local Parquet file using GDAL.

Downloading the Data

Note

Overture only maintains publicly available datasets for a maximum of 60 days (two monthly releases). As a result, the data version URLs may need to be updated by changing the release name (e.g., `2025-09-24.0`). Use the `gdal vsi list` command, shown below, to retrieve the latest available versions. See the Overture Data Retention Policy for more details.

If the specified Overture release is no longer available, you may encounter errors similar to the following:

```
ERROR 1: AdbcStatementSetSqlQuery() failed: IO Error: Unknown error for HTTP GET to '/
↪?encoding-type=url&list-type=2&prefix=release%2F2025-06-25.0%2Ftheme%3Dplaces%2Ftype
↪%3Dplace%2F'
ERROR 1: Unable to open datasource `ADBC:' with the following drivers.
```

To check which versions are currently available, use GDAL to list them with:

Bash

PowerShell

```
export AWS_NO_SIGN_REQUEST="YES"
gdal vsi list "/vsis3/overturemaps-us-west-2/release/"
```

```
$ENV:AWS_NO_SIGN_REQUEST="YES"
gdal vsi list "/vsis3/overturemaps-us-west-2/release/"
```

A few notes on the GDAL commands below:

- The `--input` option uses an empty `ADBC:` driver prefix. This may look unusual since it does not include a filename or URL; instead, all the logic is provided via the SQL `--oo` (`--open-option`) parameter.
- Each of the DuckDB Extensions must be installed before it can be loaded. These extensions are downloaded from http://extensions.duckdb.org. Once installed, you only need to provide the `LOAD` option to enable them.
- In this tutorial, we use the spatial, parquet, httpfs, and aws extensions. These are all core DuckDB extensions.

- Each `--oo PRELUDE_STATEMENTS` option must be provided separately. Unlike the Postgres GDAL driver, you cannot use a `;` separator to combine multiple statements on a single line.
- We limit results to addresses within the state of Vermont by adding `address_levels[1].value='VT'` to the SQL where clause.

gdal CLI

Traditional

PowerShell

```
gdal vector convert --format parquet --output overture-vt-addresses.parquet \
    --oo ADBC_DRIVER=duckdb \
    --oo PRELUDE_STATEMENTS="INSTALL spatial" \
    --oo PRELUDE_STATEMENTS="LOAD spatial" \
    --oo PRELUDE_STATEMENTS="INSTALL parquet" \
    --oo PRELUDE_STATEMENTS="LOAD parquet" \
    --oo PRELUDE_STATEMENTS="INSTALL httpfs" \
    --oo PRELUDE_STATEMENTS="LOAD httpfs" \
    --oo PRELUDE_STATEMENTS="INSTALL aws" \
    --oo PRELUDE_STATEMENTS="LOAD aws" \
    --oo SQL="select * from read_parquet('s3://overturemaps-us-west-2/release/2025-09-24.
↪0/theme=addresses/type=address/*') where country='US' and bbox.xmin between -74 and -
↪73 and bbox.ymin between 42.7 and 45.1 and address_levels[1].value='VT'" \
    --input ADBC: --overwrite
```

```
ogr2ogr -f parquet overture-vt-addresses.parquet \
    -oo ADBC_DRIVER=libduckdb \
    -oo PRELUDE_STATEMENTS="INSTALL spatial" \
    -oo PRELUDE_STATEMENTS="LOAD spatial" \
    -oo PRELUDE_STATEMENTS="INSTALL parquet" \
    -oo PRELUDE_STATEMENTS="LOAD parquet" \
    -oo PRELUDE_STATEMENTS="INSTALL httpfs" \
    -oo PRELUDE_STATEMENTS="LOAD httpfs" \
    -oo PRELUDE_STATEMENTS="INSTALL aws" \
    -oo PRELUDE_STATEMENTS="LOAD aws" \
    -oo SQL="select * from read_parquet('s3://overturemaps-us-west-2/release/2025-09-24.
↪0/theme=addresses/type=address/*') where country='US' and bbox.xmin between -74 and -
↪73 and bbox.ymin between 42.7 and 45.1 and address_levels[1].value='VT'" \
    ADBC:
```

```
gdal vector convert --format parquet --output overture-vt-addresses.parquet `
    --oo ADBC_DRIVER=duckdb `
    --oo PRELUDE_STATEMENTS="INSTALL spatial" `
    --oo PRELUDE_STATEMENTS="LOAD spatial" `
    --oo PRELUDE_STATEMENTS="INSTALL parquet" `
    --oo PRELUDE_STATEMENTS="LOAD parquet" `
    --oo PRELUDE_STATEMENTS="INSTALL httpfs" `
    --oo PRELUDE_STATEMENTS="LOAD httpfs" `
    --oo PRELUDE_STATEMENTS="INSTALL aws" `
    --oo PRELUDE_STATEMENTS="LOAD aws" `
    --oo SQL="select * from read_parquet('s3://overturemaps-us-west-2/release/2025-09-24.
↪0/theme=addresses/type=address/*') where country='US' and bbox.xmin between -74 and -
↪73 and bbox.ymin between 42.7 and 45.1 and address_levels[1].value='VT'" `
```

(continues on next page)

(continued from previous page)

```
    --input ADBC: --overwrite
```

Visualizing the Data

Now that we have our local Parquet file, we can output some sample records to verify its contents.

gdal CLI

Traditional

PowerShell

```
gdal vector pipeline \
    ! read overture-vt-addresses.parquet \
    ! limit 5 \
    ! select number street postcode country \
    ! write /vsistdout/ --output-format CSV
```

```
ogr2ogr -f CSV /vsistdout/ overture-vt-addresses.parquet -limit 5 -select number,street,
↪postcode,country
```

```
gdal vector pipeline \
    ! read overture-vt-addresses.parquet \
    ! limit 5 \
    ! select number street postcode country \
    ! write /vsistdout/ --output-format CSV --quiet
```

The commands above should display 5 records in a tabular format, similar to the example below:

```
country,postcode,street,number
US,"05261",CROW HILL Road,"316"
US,"05261",KRUM Road,"550"
US,"05261",KRUM Road,"643"
US,"05261",NORTHWEST HILL Road,"1744"
US,"05261",NORTHWEST HILL Road,"1763"
```

10.3.4.4 Creating Hex Bins of Point Data

In this example, we select all points of interest (POIs) in France and use the h3 DuckDB community extension to group them into hexagonal bins.

The results are saved locally to a new `POI_FR_H3.parquet` file, which can then be used with MapServer to generate a density map.

Downloading the Data

Some notes on the GDAL commands below:

- Since we installed several DuckDB extensions in the first step, we only need to `LOAD` them now; there is no need to run `INSTALL` again.
- We install a new h3 community extension, which provides hexagonal indexing functionality.

gdal CLI

Traditional

PowerShell

```
gdal vector pipeline \
    ! read --input ADBC: \
    --oo ADBC_DRIVER=libduckdb \
    --oo PRELUDE_STATEMENTS="LOAD spatial" \
    --oo PRELUDE_STATEMENTS="LOAD parquet" \
    --oo PRELUDE_STATEMENTS="LOAD httpfs" \
    --oo PRELUDE_STATEMENTS="LOAD aws" \
    --oo PRELUDE_STATEMENTS="INSTALL h3 FROM community" \
    --oo PRELUDE_STATEMENTS="LOAD h3" \
    --oo SQL="select h3_id, ST_GeomFromText(h3_cell_to_boundary_wkt(h3_id)) geometry,␣
↪count(*) POI_count \
        from (select id, h3_latlng_to_cell_string(st_y(geometry), st_x(geometry), 6) as␣
↪h3_id from \
        read_parquet('s3://overturemaps-us-west-2/release/2025-09-24.0/theme=places/
↪type=place/*') \
        where addresses[1].country='FR' and bbox.xmin between -5 and 9 and bbox.ymin␣
↪between 41 and 51) group by h3_id" \
    ! edit --crs epsg:4326 \
    ! write --format parquet POI_FR_H3.parquet
```

```
ogr2ogr -f parquet POI_FR_H3.parquet -a_srs epsg:4326 \
    -oo ADBC_DRIVER=libduckdb \
    -oo PRELUDE_STATEMENTS="LOAD spatial" \
    -oo PRELUDE_STATEMENTS="LOAD parquet" \
    -oo PRELUDE_STATEMENTS="LOAD httpfs" \
    -oo PRELUDE_STATEMENTS="LOAD aws" \
    -oo PRELUDE_STATEMENTS="INSTALL h3 FROM community" \
    -oo PRELUDE_STATEMENTS="LOAD h3" \
    -oo SQL="select h3_id, ST_GeomFromText(h3_cell_to_boundary_wkt(h3_id)) geometry,␣
↪count(*) POI_count \
        from (select id, h3_latlng_to_cell_string(st_y(geometry), st_x(geometry), 6) as␣
↪h3_id from \
        read_parquet('s3://overturemaps-us-west-2/release/2025-09-24.0/theme=places/
↪type=place/*') \
        where addresses[1].country='FR' and bbox.xmin between -5 and 9 and bbox.ymin␣
↪between 41 and 51) group by h3_id" \
    ADBC:
```

```
gdal vector pipeline `
    ! read --input ADBC: `
    --oo ADBC_DRIVER=duckdb `
    --oo PRELUDE_STATEMENTS="LOAD spatial" `
    --oo PRELUDE_STATEMENTS="LOAD parquet" `
    --oo PRELUDE_STATEMENTS="LOAD httpfs" `
    --oo PRELUDE_STATEMENTS="LOAD aws" `
    --oo PRELUDE_STATEMENTS="INSTALL h3 FROM community" `
    --oo PRELUDE_STATEMENTS="LOAD h3" `
    --oo SQL="select h3_id, ST_GeomFromText(h3_cell_to_boundary_wkt(h3_id)) geometry,␣
↪count(*) POI_count `
        from (select id, h3_latlng_to_cell_string(st_y(geometry), st_x(geometry), 6) as␣
↪h3_id from `
```

(continues on next page)

(continued from previous page)

```
        read_parquet('s3://overturemaps-us-west-2/release/2025-09-24.0/theme=places/
↪type=place/*') `
        where addresses[1].country='FR' and bbox.xmin between -5 and 9 and bbox.ymin␣
↪between 41 and 51) group by h3_id" `
    ! edit --crs epsg:4326 `
    ! write --format parquet POI_FR_H3.parquet
```

Visualizing the Data

We will use MapServer to create a simple map from the dataset we downloaded. MapServer can be installed via Conda:

```
conda install --yes -c conda-forge mapserver
```

Next, create a new Mapfile called `hexbin.map` in the same folder as the `POI_FR_H3.parquet` dataset created in the previous step, using the code provided below:

Bash

PowerShell

```
echo -e 'MAP
    NAME "HEXBIN"
    SIZE 2000 2000
    EXTENT -5.271 44.426 4.084 50.089
    PROJECTION
        "EPSG:4326"
    END
    LAYER
        NAME "POI_FR_H3"
        STATUS ON
        CONNECTION "POI_FR_H3.parquet"
        TYPE POLYGON
        CONNECTIONTYPE OGR
        CLASS
            STYLE
                RANGEITEM "POI_count"
                COLORRANGE "#FFF7BC" "#D73027"
                DATARANGE 1 1000
            END
        END
    END
END' > hexbin.map
```

```
@'
MAP
    NAME "HEXBIN"
    SIZE 2000 2000
    EXTENT -5.271 44.426 4.084 50.089
    PROJECTION
        "EPSG:4326"
    END
    LAYER
        NAME "POI_FR_H3"
```

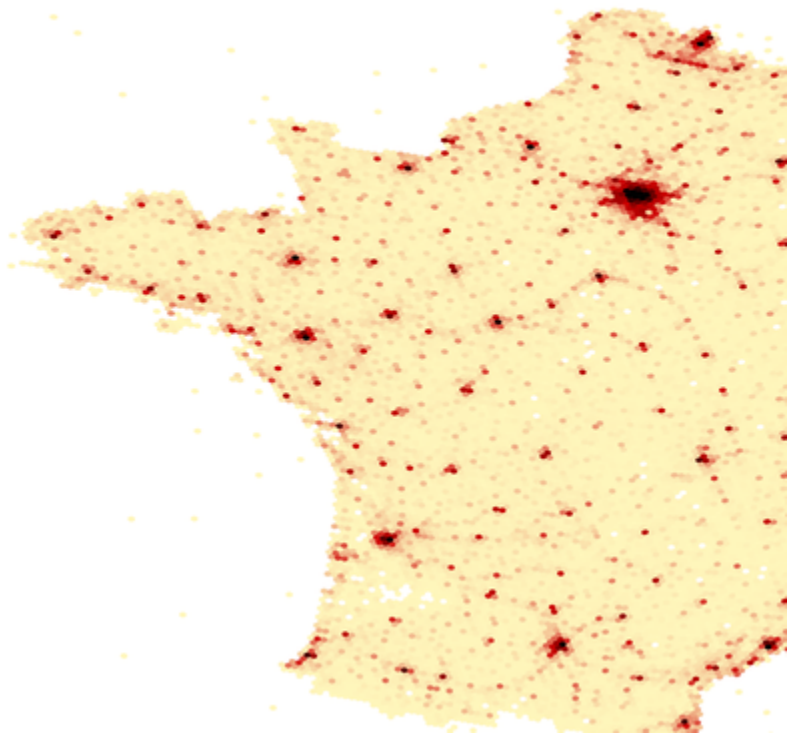
(continues on next page)

(continued from previous page)

```
        STATUS ON
        CONNECTION "POI_FR_H3.parquet"
        TYPE POLYGON
        CONNECTIONTYPE OGR
        CLASS
            STYLE
                RANGEITEM "POI_count"
                COLORRANGE "#FFF7BC" "#D73027"
                DATARANGE 1 1000
            END
        END
    END
END
'@ | Set-Content -Path "hexbin.map"
```

Finally, we can run the following command to generate the `hexbin.png` image.

```
map2img -m hexbin.map -o hexbin.png
```



10.3.5 Vector driver in Python implementation tutorial

Added in version 3.1.

10.3.5.1 Introduction

Since GDAL 3.1, the capability of writing read-only vector drivers in Python has been added. It is strongly advised to read the *Vector driver implementation tutorial* first, which will give the general principles of how a vector driver works.

This capability does not require the use of the GDAL/OGR SWIG Python bindings (but a vector Python driver may use them.)

Note: per project policies, this is considered as an "experimental" feature and the GDAL project will not accept such Python drivers to be included in the GDAL repository. Drivers aiming at inclusion in GDAL master should priorly be ported to C++. The rationale for this is that:

- the correctness of the Python code can mostly be checked at runtime, whereas C++ benefits from static analysis (at compile time, and other checkers).
- Python code is executed under the Python Global Interpreter Lock, which makes them not scale.
- Not all builds of GDAL have Python available.

10.3.5.2 Linking mechanism to a Python interpreter

See *Linking mechanism to a Python interpreter*

10.3.5.3 Driver location

Driver filenames must start with *gdal_* or *ogr_* and have the *.py* extension. They will be searched in the following directories:

- the directory pointed by the `GDAL_PYTHON_DRIVER_PATH` configuration option (there may be several paths separated by *:* on Unix or *;* on Windows)
- if not defined, the directory pointed by the `GDAL_DRIVER_PATH` configuration option.
- if not defined, in the directory (hardcoded at compilation time on Unix builds) where native plugins are located.

GDAL does not try to manage Python dependencies that are imported by the driver .py script. It is up to the user to make sure its current Python environment has all required dependencies installed.

10.3.5.4 Import section

Drivers must have the following import section to load the base classes.

```
from gdal_python_driver import BaseDriver, BaseDataset, BaseLayer
```

The `gdal_python_driver` module is created dynamically by GDAL and is not present on the file system.

10.3.5.5 Metadata section

In the first 1000 lines of the .py file, a number of required and optional KEY=VALUE driver directives must be defined. They are parsed by C++ code, without using the Python interpreter, so it is vital to respect the following constraints:

- each declaration must be on a single line, and start with `# gdal: DRIVER_` (space character between sharp character and gdal, and between colon character and DRIVER_)
- the value must be a literal value of type string (except for # gdal: DRIVER_SUPPORTED_API_VERSION which can accept an array of integers), without expressions, function calls, escape sequences, etc.
- strings may be single or double-quoted

The following directives must be declared:

- `# gdal: DRIVER_NAME = "NAME"`: the short name of the driver
- `# gdal: DRIVER_SUPPORTED_API_VERSION = [1]`: the API version(s) supported by the driver. Must include 1, which is the only currently supported version in GDAL 3.1
- `# gdal: DRIVER_DCAP_VECTOR = "YES"`: declares a vector driver
- `# gdal: DRIVER_DMD_LONGNAME = "a longer name of the driver"`

Additional directives:

- `# gdal: DRIVER_DMD_EXTENSIONS = "ext1 ext2"`: list of extension(s) recognized by the driver, without the dot, and separated by space
- `# gdal: DRIVER_DMD_HELPTOPIC = "https://example.com/my_help.html"`: URL to a help page for the driver
- `# gdal: DRIVER_DMD_OPENOPTIONLIST = "<OpenOptionList><Option name='OPT1' type='boolean' description='bla' default='NO'/></OpenOptionList>"` where the XML is an `OptionOptionList`.
- and all other metadata items found in gdal.h starting with `GDAL_DMD_` or `GDAL_DCAP` by creating an item name which starts with `# gdal: DRIVER_` and the value of the `GDAL_DMD_` or `GDAL_DCAP` metadata item. For example `#define GDAL_DMD_CONNECTION_PREFIX "DMD_CONNECTION_PREFIX"` becomes `# gdal: DRIVER_DMD_CONNECTION_PREFIX`

Example:

```
# gdal: DRIVER_NAME = "DUMMY"
# gdal: DRIVER_SUPPORTED_API_VERSION = [1]
# gdal: DRIVER_DCAP_VECTOR = "YES"
# gdal: DRIVER_DMD_LONGNAME = "my dummy plugin"
# gdal: DRIVER_DMD_EXTENSIONS = "foo bar"
# gdal: DRIVER_DMD_HELPTOPIC = "https://example.com/my_help.html"
```

10.3.5.6 Driver class

The entry point .py script must contains a single class that inherits from `gdal_python_driver.BaseDriver`.

That class must define the following methods:

identify(*self*, *filename*, *first_bytes*, *open_flags*, *open_options={}*)

Parameters

- **filename** (*str*) -- File name, or more generally, connection string.
- **first_bytes** (*bytes*) -- First bytes of the file (if it is a file). At least 1024 (if the file has at least 1024 bytes), or more if a native driver in the driver probe sequence has requested more previously.
- **open_flags** (*int*) -- Open flags. To be ignored for now.
- **open_options** (*dict*) -- Open options.

Returns

True if the file is recognized by the driver, False if not, or -1 if that cannot be known from the first bytes.

open(*self*, *filename*, *first_bytes*, *open_flags*, *open_options={}*)

Parameters

- **filename** (*str*) -- File name, or more generally, connection string.
- **first_bytes** (*bytes*) -- First bytes of the file (if it is a file). At least 1024 (if the file has at least 1024 bytes), or more if a native driver in the driver probe sequence has requested more previously.
- **open_flags** (*int*) -- Open flags. To be ignored for now.
- **open_options** (*dict*) -- Open options.

Returns
an object deriving from gdal_python_driver.BaseDataset or None

Example:

```
# Required: class deriving from BaseDriver
class Driver(BaseDriver):

    def identify(self, filename, first_bytes, open_flags, open_options={}):
        return filename == 'DUMMY:'

    # Required
    def open(self, filename, first_bytes, open_flags, open_options={}):
        if not self.identify(filename, first_bytes, open_flags):
            return None
        return Dataset(filename)
```

10.3.5.7 Dataset class

The Driver.open() method on success should return an object from a class that inherits from `gdal_python_driver.BaseDataset`.

Layers

The role of this object is to store vector layers. There are two implementation options. If the number of layers is small or they are fast to construct, then the `__init__` method can defined a `layers` attribute that is a sequence of objects from a class that inherits from `gdal_python_driver.BaseLayer`.

Example:

```
class Dataset(BaseDataset):

    def __init__(self, filename):
        self.layers = [Layer(filename)]
```

Otherwise, the following two methods should be defined:

layer_count(*self*)

Returns
the number of layers

layer(*self*, *idx*)

Parameters
idx (*int*) -- Index of the layer to return. Normally between 0 and self.layer_count() - 1, but calling code might pass any value. In case of invalid index, None should be returned.

Returns
an object deriving from gdal_python_driver.BaseLayer or None. The C++ code will take care of caching that object, and this method will only be called once for a given idx value.

Example:

```
class Dataset(BaseDataset):

    def layer_count(self):
        return 1

    def layer(self, idx):
        return [Layer(self.filename)] if idx = 0 else None
```

Metadata

The dataset may define a `metadata` dictionary, in `__init__` of key: value of type string, for the default metadata domain. Alternatively, the following method may be implemented.

metadata(*self*, *domain*)

Parameters
domain (*str*) -- metadata domain. Empty string for the default one

Returns
None, or a dictionary of key:value pairs of type string;

Other methods

The following method may be optionally implemented:

close(*self*)

Called at the destruction of the C++ peer GDALDataset object. Useful to close database connections for example.

10.3.5.8 Layer class

The Dataset object will instantiate one or several objects from a class that inherits from `gdal_python_driver.BaseLayer`.

Metadata, and other definitions

The following attributes are required and must defined at __init__ time:

name

Layer name, of type string. If not set, a `name` method must be defined.

fields

Sequence of field definitions (may be empty). Each field is a dictionary with the following properties:

name

Required

type

A integer value of type ogr.OFT_ (from the SWIG Python bindings), or one of the following string values: `String`, `Integer`, `Integer16`, `Integer64`, `Boolean`, `Real`, `Float`, `Binary`, `Date`, `Time`, `DateTime`

If that attribute is not set, a `fields` method must be defined and return such a sequence.

geometry_fields

Sequence of geometry field definitions (may be empty). Each field is a dictionary with the following properties:

name

Required. May be empty

type

Required. A integer value of type ogr.wkb_ (from the SWIG Python bindings), or one of the following string values: `Unknown`, `Point`, `LineString`, `Polygon`, `MultiPoint`, `MultiLineString`, `MultiPolygon`, `GeometryCollections` or all other values returned by *`OGRGeometryTypeToName()`*

srs

The SRS attached to the geometry field as a string that can be ingested by *`OGRSpatialReference::SetFromUserInput()`*, such as a PROJ string, WKT string, or `AUTHORITY:CODE`.

If that attribute is not set, a `geometry_fields` method must be defined and return such a sequence.

The following attributes are optional:

fid_name

Feature ID column name, of type string. May be empty string. If not set, a `fid_name` method may be defined.

metadata

A dictionary of key: value strings, corresponding to metadata of the default metadata domain. Alternatively, a `metadata` method that accepts a domain argument may be defined.

iterator_honour_attribute_filter

Can be set to True if the feature iterator takes into account the `attribute_filter` attribute that can be set on the layer.

iterator_honour_spatial_filter

Can be set to True if the feature iterator takes into account the `spatial_filter` attribute that can be set on the layer.

feature_count_honour_attribute_filter

Can be set to True if the feature_count method takes into account the `attribute_filter` attribute that can be set on the layer.

feature_count_honour_spatial_filter

Can be set to True if the feature_count method takes into account the `spatial_filter` attribute that can be set on the layer.

Feature iterator

The Layer class must implement the iterator interface, so typically with a `__iter__` method.

The resulting iterator must produce dictionaries for each feature's content. The keys allowed in the returned dictionary are:

id

Strongly recommended. The value must be an integer to be recognized as a FID.

type

Required. The value must be the string "OGRFeature"

fields

Required. The value must be either a dictionary whose keys are field names; or None

geometry_fields

Required. the value must be a dictionary whose keys are geometry field names (possibly the empty string for unnamed geometry columns); or None.

The value of each key must be either a geometry encoded as a WKT string; a geometry encoded as ISO WKB as a bytes-like object; or None.

style

Optional. The value must be a string conforming to the *Feature Style Specification*.

Filtering

By default, any attribute or spatial filter set by the user of the OGR API will be evaluated by the generic C++ side of the driver, by iterating over all features of the layer.

If the `iterator_honour_attribute_filter` (resp. `iterator_honour_spatial_filter`) attribute of the layer object is set to `True`, the attribute filter (resp. spatial filter) must be honoured by the feature iterator method.

The attribute filter is set in the `attribute_filter` attribute of the layer object. It is a string conforming to *OGR SQL*. When the attribute filter is changed by the OGR API, the `attribute_filter_changed` optional method is called (see below paragraph about optional methods). An implementation of `attribute_filter_changed` may decide to fallback on evaluation by the generic C++ side of the driver by calling the `SetAttributeFilter` method (see below passthrough example)

The geometry filter is set in the `spatial_filter` attribute of the layer object. It is a string encoding as ISO WKT. It is the responsibility of the user of the OGR API to express it in the CRS of the layer. When the attribute filter is changed by the OGR API, the `spatial_filter_changed` optional method is called (see below paragraph about optional methods). An implementation of `spatial_filter_changed` may decide to fallback on evaluation by the generic C++ side of the driver by calling the `SetSpatialFilter` method (see below passthrough example)

Optional methods

The following methods may be optionally implemented:

extent(*self*, *force_computation*)

Returns
the list [xmin,ymin,xmax,ymax] with the spatial extent of the layer.

feature_count(*self*, *force_computation*)

Returns
the number of features of the layer.

If self.feature_count_honour_attribute_filter or self.feature_count_honour_spatial_filter are set to True, the attribute filter and/or spatial filter must be honoured by this method.

feature_by_id(*self*, *fid*)

Parameters
fid (*int*) -- feature ID

Returns
a feature object in one of the formats of the `__next__` method described above, or None if no object matches fid

`attribute_filter_changed`(*self*)

This method is called whenever self.attribute_filter has been changed. It is the opportunity for the driver to potentially change the value of self.iterator_honour_attribute_filter or feature_count_honour_attribute_filter attributes.

`spatial_filter_changed`(*self*)

This method is called whenever self.spatial_filter has been changed (its value is a geometry encoded in WKT) It is the opportunity for the driver to potentially change the value of self.iterator_honour_spatial_filter or feature_count_honour_spatial_filter attributes.

`test_capability`(*self*, *cap*)

Parameters
cap (*string*) -- potential values are BaseLayer.FastGetExtent, BaseLayer.FastSpatialFilter, BaseLayer.FastFeatureCount, BaseLayer.RandomRead, BaseLayer.StringsAsUTF8 or other strings supported by *`OGRLayer::TestCapability()`*

Returns
True if the capability is supported, False otherwise.

10.3.5.9 Full example

The following example is a passthrough driver that forwards the calls to the SWIG Python GDAL API. It has no practical use, and is just intended to show case most possible uses of the API. A real-world driver will only use part of the API demonstrated. For example, the passthrough driver implements attribute and spatial filters in a completely dummy way, by calling back the C++ part of the driver. The `iterator_honour_attribute_filter` and `iterator_honour_spatial_filter` attributes, and the `attribute_filter_changed` and `spatial_filter_changed` method implementations, could have omitted with the same result.

The connection strings recognized by the drivers are `PASSHTROUGH:connection_string_supported_by_non_python_drivers`. Note that the prefixing by the driver name is absolutely not a requirement, but something specific to this particular driver which is a bit artificial (without the prefix, the connection string would go directly to the native driver). The CityJSON driver mentioned in the *Other examples* paragraph does not need it.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This code is in the public domain, so as to serve as a template for
# real-world plugins.
# or, at the choice of the licensee,
# Copyright 2019 Even Rouault
# SPDX-License-Identifier: MIT

# gdal: DRIVER_NAME = "PASSTHROUGH"
# API version(s) supported. Must include 1 currently
# gdal: DRIVER_SUPPORTED_API_VERSION = [1]
# gdal: DRIVER_DCAP_VECTOR = "YES"
# gdal: DRIVER_DMD_LONGNAME = "Passthrough driver"
# gdal: DRIVER_DMD_CONNECTION_PREFIX = "PASSTHROUGH:"

from osgeo import gdal, ogr
```

(continues on next page)

(continued from previous page)

```
from gdal_python_driver import BaseDriver, BaseDataset, BaseLayer

class Layer(BaseLayer):

    def __init__(self, gdal_layer):
        self.gdal_layer = gdal_layer
        self.name = gdal_layer.GetName()
        self.fid_name = gdal_layer.GetFIDColumn()
        self.metadata = gdal_layer.GetMetadata_Dict()
        self.iterator_honour_attribute_filter = True
        self.iterator_honour_spatial_filter = True
        self.feature_count_honour_attribute_filter = True
        self.feature_count_honour_spatial_filter = True

    def fields(self):
        res = []
        layer_defn = self.gdal_layer.GetLayerDefn()
        for i in range(layer_defn.GetFieldCount()):
            ogr_field_def = layer_defn.GetFieldDefn(i)
            field_def = {"name": ogr_field_def.GetName(),
                         "type": ogr_field_def.GetType()}
            res.append(field_def)
        return res

    def geometry_fields(self):
        res = []
        layer_defn = self.gdal_layer.GetLayerDefn()
        for i in range(layer_defn.GetGeomFieldCount()):
            ogr_field_def = layer_defn.GetGeomFieldDefn(i)
            field_def = {"name": ogr_field_def.GetName(),
                         "type": ogr_field_def.GetType()}
            srs = ogr_field_def.GetSpatialRef()
            if srs:
                field_def["srs"] = srs.ExportToWkt()
            res.append(field_def)
        return res

    def test_capability(self, cap):
        if cap in (BaseLayer.FastGetExtent, BaseLayer.StringsAsUTF8,
                BaseLayer.RandomRead, BaseLayer.FastFeatureCount):
            return self.gdal_layer.TestCapability(cap)
        return False

    def extent(self, force_computation):
        # Impedance mismatch between SWIG GetExtent() and the Python
        # driver API
        minx, maxx, miny, maxy = self.gdal_layer.GetExtent(force_computation)
        return [minx, miny, maxx, maxy]

    def feature_count(self, force_computation):
        # Dummy implementation: we call back the generic C++ implementation
        return self.gdal_layer.GetFeatureCount(True)
```

(continues on next page)

(continued from previous page)

```
    def attribute_filter_changed(self):
        # Dummy implementation: we call back the generic C++ implementation
        if self.attribute_filter:
            self.gdal_layer.SetAttributeFilter(str(self.attribute_filter))
        else:
            self.gdal_layer.SetAttributeFilter(None)

    def spatial_filter_changed(self):
        # Dummy implementation: we call back the generic C++ implementation
        # the 'inf' test is just for a test_ogrsf oddity
        if self.spatial_filter and 'inf' not in self.spatial_filter:
            self.gdal_layer.SetSpatialFilter(
                ogr.CreateGeometryFromWkt(self.spatial_filter))
        else:
            self.gdal_layer.SetSpatialFilter(None)

    def _translate_feature(self, ogr_f):
        fields = {}
        layer_defn = ogr_f.GetDefnRef()
        for i in range(ogr_f.GetFieldCount()):
            if ogr_f.IsFieldSet(i):
                fields[layer_defn.GetFieldDefn(i).GetName()] = ogr_f.GetField(i)
        geom_fields = {}
        for i in range(ogr_f.GetGeomFieldCount()):
            g = ogr_f.GetGeomFieldRef(i)
            if g:
                geom_fields[layer_defn.GetGeomFieldDefn(
                    i).GetName()] = g.ExportToIsoWKb()
        return {'id': ogr_f.GetFID(),
                'type': 'OGRFeature',
                'style': ogr_f.GetStyleString(),
                'fields': fields,
                'geometry_fields': geom_fields}

    def __iter__(self):
        for f in self.gdal_layer:
            yield self._translate_feature(f)

    def feature_by_id(self, fid):
        ogr_f = self.gdal_layer.GetFeature(fid)
        if not ogr_f:
            return None
        return self._translate_feature(ogr_f)

class Dataset(BaseDataset):

    def __init__(self, gdal_ds):
        self.gdal_ds = gdal_ds
        self.layers = [Layer(gdal_ds.GetLayer(idx))
                       for idx in range(gdal_ds.GetLayerCount())]
        self.metadata = gdal_ds.GetMetadata_Dict()
```

(continues on next page)

(continued from previous page)

```
    def close(self):
        del self.gdal_ds
        self.gdal_ds = None


class Driver(BaseDriver):

    def _identify(self, filename):
        prefix = 'PASSTHROUGH:'
        if not filename.startswith(prefix):
            return None
        return gdal.OpenEx(filename[len(prefix):], gdal.OF_VECTOR)

    def identify(self, filename, first_bytes, open_flags, open_options={}):
        return self._identify(filename) is not None

    def open(self, filename, first_bytes, open_flags, open_options={}):
        gdal_ds = self._identify(filename)
        if not gdal_ds:
            return None
        return Dataset(gdal_ds)
```

10.3.5.10 Other examples

Other examples, including a CityJSON driver, may be found at https://github.com/OSGeo/gdal/tree/master/examples/pydrivers

10.4 Geographic Network Model

10.4.1 GNM API tutorial

This document is intended to describe using the GNM C++ classes to work with networks. It is advised to read the *Geographic Networks Data Model* before to understand the purpose and structure of GNM classes.

10.4.1.1 Managing networks

In the first example we will create a small water network on the base of the set of spatial data (two shapefiles: pipes and wells which are situated at the GDAL source tree: autotest\gnm\data). The use of the common network format - `GNMGdalNetwork` class - will allow us to select one of the GDAL-supported vector formats for our network - ESRI Shapefile. After the creation we will build a topology and add some additional data: pumps layer, in order to manually edit network topology.

Initially we register GDAL drivers and create some options (string pairs), which will be passed as parameters during network creation. Here we create a network's name.

```
#include "gnm.h"
#include <vector>

int main ()
{
    GDALAllRegister();
```

(continues on next page)

(continued from previous page)

```
    char **papszDSCO = NULL;
    papszDSCO = CSLAddNameValue(papszDSCO, GNM_MD_NAME, "my_pipes_network");
    papszDSCO = CSLAddNameValue(papszDSCO, GNM_MD_SRS, "EPSG:4326");
    papszDSCO = CSLAddNameValue(papszDSCO, GNM_MD_DESCR, "My pipes network");
    papszDSCO = CSLAddNameValue(papszDSCO, GNM_MD_FORMAT, "ESRI Shapefile");
```

Some options are obligatory. The following parameters must be specified during the network creation: the path/name; format of network storage; spatial reference system (EPSG, WKT, etc.). The according dataset with the "network part" will be created and the resulting network will be returned.

```
GDALDriver *poDriver = GetGDALDriverManager()->GetDriverByName("GNMFile");
GNMGenericNetwork* poDS = (GNMGenericNetwork*) poDriver->Create( "..\\network_data", 0,␣
↪0, 0, GDT_Unknown,
                                                        papszDSCO );
CSLDestroy(papszDSCO);
```

For now we have a void network consisted of only "system layers". We need to populate it with "class layers" full of features, so we open a certain foreign dataset and copy layers from it to our network. Note, that we use GDALDataset:: methods for working with "class layers", because *GNMNetwork* inherited from GDALDataset.

```
GDALDataset *poSrcDS = (GDALDataset*) GDALOpenEx("..\\in_data",
                                GDAL_OF_VECTOR | GDAL_OF_READONLY, NULL, NULL, NULL );

OGRLayer *poSrcLayer1 = poSrcDS->GetLayerByName("pipes");
OGRLayer *poSrcLayer2 = poSrcDS->GetLayerByName("wells");

poDS->CopyLayer(poSrcLayer1, "pipes");
poDS->CopyLayer(poSrcLayer2, "wells");

GDALClose(poSrcDS);
```

After the successful copying we have the network full of features, but with no topology. The features were added and registered in the network but they are still not connected with each other. Now it is time to build the network topology. There are two ways of doing this in GNM: manually or automatically. In the most cases automatic building is more convenient, while manual is useful for small edits. Automatic building requires some parameters: we must specify which "class layers" will participate in topology building (we select our two layers), a snap tolerance, direct and inverse cost, direction, which is equal 0.00005 in our case. If the building will be successful the network's graph will be filled with the according connections.

```
printf("\nBuilding network topology ...\n");
char **papszLayers = NULL;
for(int i = 0; i < poDS->GetLayerCount(); ++i)
{
    OGRLayer* poLayer = poDS->GetLayer(i);
    papszLayers = CSLAddString(papszLayers, poLayer->GetName() );
}

if(poGenericNetwork->ConnectPointsByLines(papszLayers, dfTolerance,
                                  dfDirCost, dfInvCost, eDir) != CE_None )
{
    printf("Building topology failed\n");
}
```

(continues on next page)

(continued from previous page)

```
else
{
    printf("Topology has been built successfully\n");
}
```

At this point we have a ready network with topological and spatial data, which can be used now for different purposes (analysis, converting into different formats, etc). But sometimes it is necessary to modify some network's data. For example we need to add additional features and attach them to our built topology (modify topology). We create a new "class layer" in the network and add one feature to it.

```
OGRLayer *poNewLayer = poDS->CreateLayer("pumps", , NULL, wkbPoint, NULL );
if( poNewLayer == NULL )
{
    printf( "Layer creation failed.\n" );
    exit( 1 );
}

OGRFieldDefn fieldDefn ("pressure",OFTReal);
if( poNewLayer->CreateField( &fieldDefn ) != OGRERR_NONE )
{
    printf( "Creating Name field failed.\n" );
    exit( 1 );
}

OGRFeature *poFeature = OGRFeature::CreateFeature(poNewLayer->GetLayerDefn());
OGRPoint pt;
pt.setX(37.291466);
pt.setY(55.828351);
poFeature->SetGeometry(&pt);
if( poNewLayer->CreateFeature( poFeature ) != OGRERR_NONE )
{
    printf( "Failed to create feature.\n" );
    exit( 1 );
}

GNMGFID gfid = poFeature->GetFID();

OGRFeature::DestroyFeature( poFeature );
```

After the successful creation the feature will be registered in the network and we can connect it with others. There can be two possible ways to do this. In the first case we need a real feature which will be an edge in the connection, while in the second case we do not need such feature, and passing -1 into the *GNMGenericNetwork::ConnectFeatures()* method means that the special system edge will be created for this connection and added to the graph automatically. In our case we had added only one point feature and we have not got the line one to be an edge, so we will use the "virtual" connection. We pass the GFID of our point as the source, the GFID of one of the existed features as the target and -1 as the connector. Note that we also set the costs (direct and inverse) and the direction of our edge manually and these values will be written to the graph. When we used the automatic connection (which also uses ConnectFeatures() internally) such vales were set automatically according to the rule which we also set before.

```
if (poDS->ConnectFeatures(gfid ,63, -1, 5.0, 5.0, GNMDirection_SrcToTgt) != GNMError_
↪None)
{
```

(continues on next page)

(continued from previous page)

```
    printf("Can not connect features\n");
}
```

After all we correctly close the network which frees the allocated resources.

```
GDALClose(poDS);
```

All in one block:

```
#include "gnm.h"
#include "gnm_priv.h"

int main ()
{
    GDALAllRegister();

    char **papszDSCO = NULL;
    papszDSCO = CSLAddNameValue(papszDSCO, GNM_MD_NAME, "my_pipes_network");
    papszDSCO = CSLAddNameValue(papszDSCO, GNM_MD_SRS, "EPSG:4326");
    papszDSCO = CSLAddNameValue(papszDSCO, GNM_MD_DESCR, "My pipes network");
    papszDSCO = CSLAddNameValue(papszDSCO, GNM_MD_FORMAT, "ESRI Shapefile");


    GDALDriver *poDriver = GetGDALDriverManager()->GetDriverByName("GNMFile");
    GNMGenericNetwork* poDS = (GNMGenericNetwork*) poDriver->Create( "..\\network_data",␣
↪0, 0, 0, GDT_Unknown,
                                                    papszDSCO );
    CSLDestroy(papszDSCO);
    if (poDS == NULL)
    {
        printf("Failed to create network\n");
        exit(1);
    }

    GDALDataset *poSrcDS = (GDALDataset*) GDALOpenEx("..\\in_data",GDAL_OF_VECTOR | GDAL_
↪OF_READONLY, NULL, NULL, NULL );
    if(poSrcDS == NULL)
    {
        printf("Can not open source dataset at\n");
        exit(1);
    }

    OGRLayer *poSrcLayer1 = poSrcDS->GetLayerByName("pipes");
    OGRLayer *poSrcLayer2 = poSrcDS->GetLayerByName("wells");
    if (poSrcLayer1 == NULL || poSrcLayer2 == NULL)
    {
        printf("Can not process layers of source dataset\n");
        exit(1);
    }

    poDS->CopyLayer(poSrcLayer1, "pipes");
    poDS->CopyLayer(poSrcLayer2, "wells");
```

(continues on next page)

(continued from previous page)

```
    GDALClose(poSrcDS);

    printf("\nBuilding network topology ...\n");
    char **papszLayers = NULL;
    for(int i = 0; i < poDS->GetLayerCount(); ++i)
    {
        OGRLayer* poLayer = poDS->GetLayer(i);
        papszLayers = CSLAddString(papszLayers, poLayer->GetName() );
    }

    if(poGenericNetwork->ConnectPointsByLines(papszLayers, dfTolerance,
                                        dfDirCost, dfInvCost, eDir) != CE_None )
    {
        printf("Building topology failed\n");
        exit(1);
    }
    else
    {
        printf("Topology has been built successfully\n");
    }

    OGRLayer *poNewLayer = poDS->CreateLayer("pumps", , NULL, wkbPoint, NULL );
    if( poNewLayer == NULL )
    {
        printf( "Layer creation failed.\n" );
        exit( 1 );
    }

    OGRFieldDefn fieldDefn ("pressure",OFTReal);
    if( poNewLayer->CreateField( &fieldDefn ) != OGRERR_NONE )
    {
        printf( "Creating Name field failed.\n" );
        exit( 1 );
    }

    OGRFeature *poFeature = OGRFeature::CreateFeature(poNewLayer->GetLayerDefn());
    OGRPoint pt;
    pt.setX(37.291466);
    pt.setY(55.828351);
    poFeature->SetGeometry(&pt);
    if( poNewLayer->CreateFeature( poFeature ) != OGRERR_NONE )
    {
        printf( "Failed to create feature.\n" );
        exit( 1 );
    }

    GNMGFID gfid = poFeature->GetFID();

    OGRFeature::DestroyFeature( poFeature );

    if (poDS->ConnectFeatures(gfid ,63, -1, 5.0, 5.0, GNMDirection_SrcToTgt) != GNMError_
```

(continues on next page)

(continued from previous page)

```
↪None)
    {
        printf("Can not connect features\n");
    }

    GDALClose(poDS);
}
```

10.4.1.2 Analysing networks

In the second example we will analyse the network which we have built in the first example. We will calculate the shortest path between two points via Dijkstra algorithm performing the feature blockings and saving the resulting path into the file.

Initially we open our network, passing the path to its Shapefile dataset.

```
#include "gnm.h"
#include "gnm_priv.h"

int main ()
{
    GDALAllRegister();

    GNMGenericNetwork *poNet = (GNMGenericNetwork*) GDALOpenEx("..\\network_data",GDAL_
↪OF_GNM | GDAL_OF_UPDATE, NULL, NULL, NULL );
    if(poSrcDS == NULL)
    {
        printf("Can not open source dataset at\n");
        exit(1);
    }
```

Before any calculations we open the dataset which will hold the layer with the resulting path.

```
GDALDataset *poResDS;
poResDS = (GDALDataset*) GDALOpenEx("..\\out_data",
                                    GDAL_OF_VECTOR | GDAL_OF_UPDATE,
                                    NULL, NULL, NULL);
if (poResDS == NULL)
{
    printf("Failed to open resulting dataset\n");
    exit(1);
}
```

Finally we use the Dijkstra shortest path method to calculations. This path will be found passing over the blocked feature and saved into internal memory OGRLayer, which we copy to the real dataset. Now it can be visualized by GIS.

```
    OGRLayer *poResLayer = poNet->GetPath(64, 41, GATDijkstraShortestPath, NULL);
    if (poResLayer == NULL)
    {
        printf("Failed to save or calculate path\n");
    }
    else if (poResDS->CopyLayer(poResLayer, "shp_tutorial.shp") == NULL)
    {
```

(continues on next page)

(continued from previous page)

```
        printf("Failed to save path to the layer\n");
    }
    else
    {
        printf("Path saved successfully\n");
    }

    GDALClose(poResDS);
    poNet->ReleaseResultSet(poRout);
    GDALClose(poNet);
}
```

All in one block:

```
#include "gnm.h"
#include "gnmstdanalysis.h"

int main ()
{
    GDALAllRegister();

    GNMGenericNetwork *poNet = (GNMGenericNetwork*) GDALOpenEx("..\\network_data",
                                                    GDAL_OF_GNM | GDAL_OF_UPDATE,
                                                    NULL, NULL, NULL );
    if(poSrcDS == NULL)
    {
        printf("Can not open source dataset at\n");
        exit(1);
    }

    GDALDataset *poResDS;
    poResDS = (GDALDataset*) GDALOpenEx("..\\out_data",
                                        GDAL_OF_VECTOR | GDAL_OF_UPDATE,
                                        NULL, NULL, NULL);
    if (poResDS == NULL)
    {
        printf("Failed to open resulting dataset\n");
        exit(1);
    }

    poNet->ChangeBlockState(36, true);

    OGRLayer *poResLayer = poNet->GetPath(64, 41, GATDijkstraShortestPath, NULL);
    if (poResLayer == NULL)
    {
        printf("Failed to save or calculate path\n");
    }
    else if (poResDS->CopyLayer(poResLayer, "shp_tutorial.shp") == NULL)
    {
        printf("Failed to save path to the layer\n");
    }
    else
```

(continues on next page)

(continued from previous page)

```
    {
        printf("Path saved successfully\n");
    }

    GDALClose(poResDS);
    poNet->ReleaseResultSet(poRout);
    GDALClose(poNet);
}
```

10.5 Projections / CRS

10.5.1 Reprojecting Data with GDAL

Reprojecting spatial data is a common task in GIS. Coordinate reference system (CRS) transformations are handled by PROJ - a required dependency of GDAL. In this tutorial we'll walk through reprojecting both vector and raster data using data from the Natural Earth dataset.

10.5.1.1 Introduction

We'll start by getting projection information from one of the Natural Earth rasters.

```
gdal raster info NE2_50M_SR.tif
```

By default, *gdal raster info* returns a summary of the CRS in a human-readable format. The CRS information in the output is shown below:

```
Coordinate Reference System:
  - name: WGS 84
  - ID: EPSG:4326
  - type: Geographic 2D
  - area of use: World, west -180.00, south -90.00, east 180.00, north 90.00
Data axis to CRS axis mapping: 2,1
```

These key details are explained below:

- **name: WGS 84** - the name of the CRS. This is a human-readable name, and is not guaranteed to be unique.
- **ID: EPSG:4326** - the unique identifier for the CRS in the EPSG registry.
- **type: Geographic 2D** - this indicates that the CRS is a geographic coordinate system (latitude and longitude), rather than a projected coordinate system (e.g. Web Mercator, UTM). The "2D" indicates that the CRS has two dimensions (latitude and longitude), as opposed to a 3D CRS which would include a vertical component (e.g. height).
- **area of use: World, west -180.00, south -90.00, east 180.00, north 90.00** - this indicates the area of the world where the CRS is intended to be used. In this case, it covers the entire world.
- **Data axis to CRS axis mapping: 2,1** - this is a GDAL specific line showing how raster data axes map to the CRS axes. Here, the raster uses (longitude, latitude) order, while the CRS defines axes as (latitude, longitude), so GDAL maps them accordingly.

We can get more detailed information about the CRS by providing the `--crs-format` option.

```
gdal raster info NE2_50M_SR.tif --crs-format=WKT2
```

This shows the definition of the CRS in the WKT2 (Well-Known Text version 2). This format was defined by the OGC and represents a complete CRS definition, not just a projection. It includes everything GDAL needs to perform coordinate transformations - a full datum definition, the projection method and parameters (for projected CRSs), and axis definitions. The full specification is found at http://www.opengis.net/doc/is/crs-wkt/2.1.11, and the current widely adopted version is ISO 19162:2019, which GDAL supports.

A breakdown of the WKT2 CRS contents is provided below.

```
Coordinate System is:
GEOGCRS["WGS 84",
    ENSEMBLE["World Geodetic System 1984 ensemble",
        MEMBER["World Geodetic System 1984 (Transit)"],
        MEMBER["World Geodetic System 1984 (G730)"],
        MEMBER["World Geodetic System 1984 (G873)"],
        MEMBER["World Geodetic System 1984 (G1150)"],
        MEMBER["World Geodetic System 1984 (G1674)"],
        MEMBER["World Geodetic System 1984 (G1762)"],
        MEMBER["World Geodetic System 1984 (G2139)"],
        MEMBER["World Geodetic System 1984 (G2296)"],
        ELLIPSOID["WGS 84",6378137,298.257223563,
            LENGTHUNIT["metre",1]],
        ENSEMBLEACCURACY[2.0]],
    PRIMEM["Greenwich",0,
        ANGLEUNIT["degree",0.0174532925199433]],
    CS[ellipsoidal,2],
        AXIS["geodetic latitude (Lat)",north,
            ORDER[1],
            ANGLEUNIT["degree",0.0174532925199433]],
        AXIS["geodetic longitude (Lon)",east,
            ORDER[2],
            ANGLEUNIT["degree",0.0174532925199433]],
    USAGE[
        SCOPE["Horizontal component of 3D system."],
        AREA["World."],
        BBOX[-90,-180,90,180]],
    ID["EPSG",4326]]
Data axis to CRS axis mapping: 2,1
```

- `GEOGCRS["WGS 84"` This means it's a Geographic CRS (latitude and longitude, not a projected CRS like Web Mercator or UTM), and has the name "WGS 84".
- `ENSEMBLE["World Geodetic System 1984 ensemble",..` - WGS 84 has been updated multiple times. The ENSEMBLE list contains all the different versions.
- `ENSEMBLEACCURACY[2.0]` means all the different versions are considered equivalent within ~2 metres.
- `ELLIPSOID["WGS 84", 6378137, 298.257223563]` defines the Earth's shape.
- `PRIMEM["Greenwich",0...` states the Prime meridian is at Greenwich (0° longitude).
- `CS[ellipsoidal,2]`, the Coordinate System is a 2-dimensional ellipsoidal.
- `AXIS["geodetic longitude (Lon)",east, ORDER[2]`, details the axis order. In this case the CRS defines axis order as latitude, longitude. This differs from the traditional GIS display order (longitude, latitude), which can lead to confusion - for more discussion around this see *CRS and axis order*.
- `USAGE[...` contains metadata associated with the coordinate system.

- `ID["EPSG",4326]` - this is the familiar **EPSG:4326** code used to reference the CRS. This identifier links the WKT definition to the EPSG registry, but the WKT provides a fully expanded CRS definition that GDAL can use directly for coordinate transformations.

You can look up the details for each coordinate system at https://spatialreference.org, for example https://spatialreference.org/ref/epsg/4326/. These pages also show coordinate systems in other formats. Alternatively, GDAL includes *gdalsrsinfo*, a command-line tool to display and validate a CRS:

```
gdalsrsinfo -V EPSG:27561
```

GDAL can work with the many different CRS formats supported by PROJ, see *Reproject data using various CRS formats*. See the full list of options at the projinfo page.

You can also use `--output-crs=PROJJSON` to return the CRS definition in PROJJSON format, which is a JSON-based format defined by PROJ for representing coordinate reference systems. This is generally easier to parse programmatically than WKT, and is also the format used when outputting dataset details in JSON (e.g. with `--format=JSON`). See *Return the CRS used by the raster using `jq`* for an example of parsing CRS information from JSON output.

```
gdal raster info NE2_50M_SR.tif --crs-format=PROJJSON
gdal raster info NE2_50M_SR.tif --format=JSON
```

Now let's reproject the raster dataset to Web Mercator, and look at the CRS information.

```
gdal raster reproject --dst-crs=EPSG:3857 NE2_50M_SR.tif NE2_50M_SR_3857.tif
gdal raster info NE2_50M_SR_3857.tif
```

Note

If you are using `bash` you can take advantage of auto-completion for CRS codes, see *Bash completion for gdal*. This suggests potential CRS codes, but taking advantage of this requires the input dataset to be specified first:

```
$ gdal raster reproject NE2_50M_SR.tif --dst-crs=EPSG:385<TAB>
3850 -- ETRS89-SWE [SWEREF 99] / RT90 5 gon O emulation  3852 -- RSRGD2000 /␣
↪DGLC2000    3855 -- EGM2008 height
3851 -- NZGD2000 / NZCS2000                              3854 -- County ST74␣
↪            3857 -- WGS 84 / Pseudo-Mercator
```

As this example uses a global raster, many CRS options are shown. When working with a more localized dataset, auto-completion will suggest only CRSs relevant to the area covered by the data.

The CRS information from the above command is shown below.

```
Coordinate Reference System:
  - name: WGS 84 / Pseudo-Mercator
  - ID: EPSG:3857
  - type: Projected
  - projection type: Popular Visualisation Pseudo Mercator
  - units: metre
  - area of use: World between 85.06°S and 85.06°N, west -180.00, south -85.06, east 180.
↪00, north 85.06
Data axis to CRS axis mapping: 1,2
```

The key differences to note here are:

- **type: Projected** - this indicates that the CRS is a projected coordinate system, which uses a map projection to convert geographic coordinates (latitude and longitude) into planar coordinates (easting and northing).
- **projection type: Popular Visualisation Pseudo Mercator** - this shows both the projection conversion name and method name. In this case the names are identical, but in some cases they may differ (e.g. the conversion name may be "UTM zone 33N" while the method is "Transverse Mercator").
- **units: metre** - the units of the projected coordinates are in metres, rather than degrees as in the original geographic CRS.
- **Data axis to CRS axis mapping: 1,2** - this indicates that the data axis order matches the CRS axis order (X, Y). In contrast, the original geographic CRS (EPSG:4326) defines axes as (latitude, longitude), requiring a mapping of 2,1.

Let's also look at the full WKT definition for this CRS.

```
gdal raster info NE2_50M_SR_3857.tif  --crs-format=WKT2
```

The CRS output is shown below.

```
Layer SRS WKT:
PROJCRS["WGS 84 / Pseudo-Mercator",
    BASEGEOGCRS["WGS 84",
        ENSEMBLE["World Geodetic System 1984 ensemble",
            MEMBER["World Geodetic System 1984 (Transit)"],
            MEMBER["World Geodetic System 1984 (G730)"],
            MEMBER["World Geodetic System 1984 (G873)"],
            MEMBER["World Geodetic System 1984 (G1150)"],
            MEMBER["World Geodetic System 1984 (G1674)"],
            MEMBER["World Geodetic System 1984 (G1762)"],
            MEMBER["World Geodetic System 1984 (G2139)"],
            MEMBER["World Geodetic System 1984 (G2296)"],
            ELLIPSOID["WGS 84",6378137,298.257223563,
                LENGTHUNIT["metre",1]],
            ENSEMBLEACCURACY[2.0]],
        PRIMEM["Greenwich",0,
            ANGLEUNIT["degree",0.0174532925199433]],
        ID["EPSG",4326]],
    CONVERSION["Popular Visualisation Pseudo-Mercator",
        METHOD["Popular Visualisation Pseudo Mercator",
            ID["EPSG",1024]],
        PARAMETER["Latitude of natural origin",0,
            ANGLEUNIT["degree",0.0174532925199433],
            ID["EPSG",8801]],
        PARAMETER["Longitude of natural origin",0,
            ANGLEUNIT["degree",0.0174532925199433],
            ID["EPSG",8802]],
        PARAMETER["False easting",0,
            LENGTHUNIT["metre",1],
            ID["EPSG",8806]],
        PARAMETER["False northing",0,
            LENGTHUNIT["metre",1],
            ID["EPSG",8807]]],
    CS[Cartesian,2],
        AXIS["easting (X)",east,
            ORDER[1],
```

(continues on next page)

(continued from previous page)

```
            LENGTHUNIT["metre",1]],
        AXIS["northing (Y)",north,
            ORDER[2],
            LENGTHUNIT["metre",1]],
    USAGE[
        SCOPE["Web mapping and visualisation."],
        AREA["World between 85.06°S and 85.06°N."],
        BBOX[-85.06,-180,85.06,180]],
    ID["EPSG",3857]]
Data axis to CRS axis mapping: 1,2
```

- `PROJCRS["WGS 84 / Pseudo-Mercator",` - this is a *projected* CRS rather than a *geographic* CRS.
- `BASEGEOGCRS["WGS 84", ...]` - the underlying coordinate reference system is based on WGS 84. The same properties as above are listed here.
- `CONVERSION["Popular Visualisation Pseudo-Mercator", ...]` this details the map projection itself.
- `CS[Cartesian,2]` - the projected units are a 2D Cartesian system (X, Y).
- `AXIS["easting (X)",east] AXIS["northing (Y)",north]` - X = easting, Y = northing.
- `LENGTHUNIT["metre",1]` - units are in metres.
- `USAGE[` - the metadata indicating the CRS is intended for web mapping, and also that the `BBOX` does not cover the full extent of WGS84 and cannot represent data at the poles.

In practice, you will often refer to a CRS using an EPSG code (e.g. EPSG:4326), but GDAL internally works with full WKT definitions like the one shown above. Next, we'll use this CRS information to reproject both raster and vector datasets.

10.5.1.2 Raster

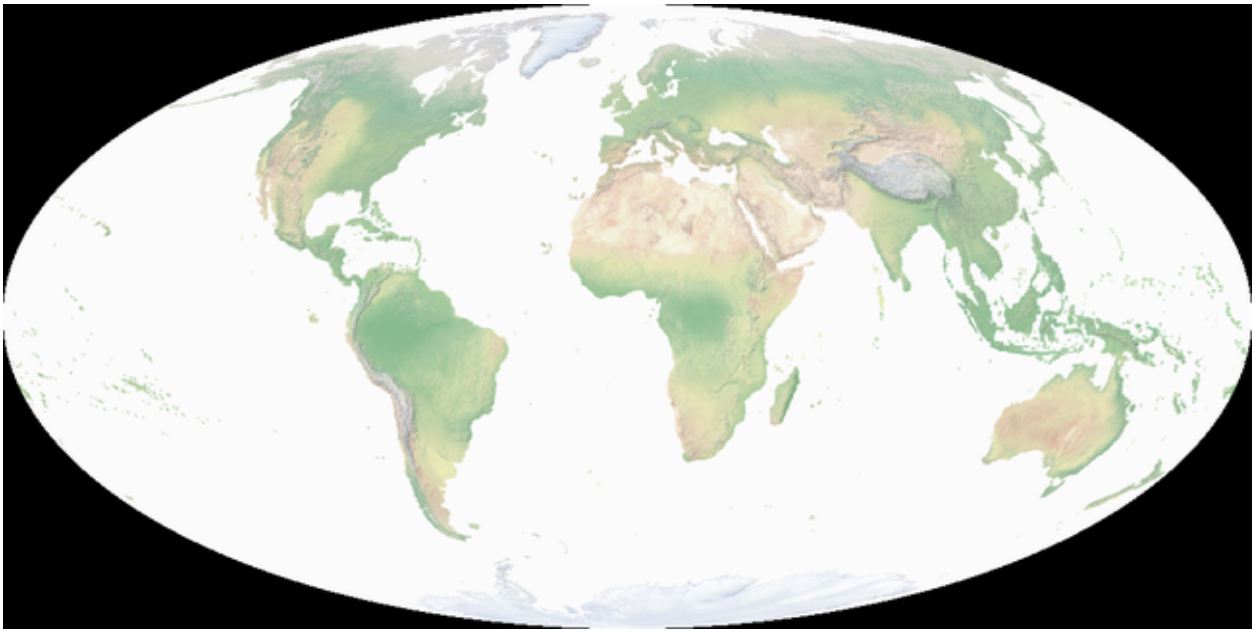
Reprojecting raster data is handled by *gdal raster reproject* (see this page for further examples). Let's try an example using Natural Earth raster data. The original data is in WGS 84 (EPSG:4326) and we'll reproject it to a CRS defined outside the EPSG registry, ESRI:53009 - the Mollweide projection. As this is a large file we'll also resize it to 10% of the original size to speed up the reprojection. The order of operations in the pipeline matters, and in this case we want to resize before reprojecting - this changes the processing time from a few minutes to a few seconds, but also reduces the output resolution and may affect reprojection accuracy.

Bash

PowerShell

```bash
gdal raster pipeline \
! read NE2_50M_SR.tif \
! resize --size=10%,10% -r average \
! reproject --dst-crs ESRI:53009 \
! write NE2_50M_SR_53009.png --overwrite
```

```powershell
gdal raster pipeline `
! read NE2_50M_SR.tif `
! resize --size=10%,10% -r average `
! reproject --dst-crs ESRI:53009 `
! write NE2_50M_SR_53009.png --overwrite
```



Note that the output is a PNG file, and the CRS information is stored in a companion file with the same name, using the extension `.aux.xml`. GDAL refers to these as PAM (Persistent Auxiliary Metadata) files. They are used to store metadata that cannot be stored in the file itself, such as the CRS for formats that do not support it (e.g. PNG, JPEG).

10.5.1.3 Vector

Reprojecting vector data is handled by *gdal vector reproject*. We can reproject the entire Natural Earth vector dataset to Web Mercator with the following command:

```
gdal vector reproject --dst-crs=EPSG:3857 natural_earth_vector.gpkg natural_earth_vector_
↪3857.gpkg --overwrite
```

If you want to reproject just a single layer, you can specify the layer name with the `--layer` option. For example, to reproject just the `ne_10m_admin_0_countries` layer:

```
gdal vector reproject --dst-crs=EPSG:3857 --layer=ne_10m_admin_0_countries natural_earth_
↪vector.gpkg natural_earth_vector_3857.gpkg --overwrite
```

If your data is already in the correct CRS, but is missing metadata, you can use *gdal vector edit* to assign the CRS information without reprojecting the data. For example, if you have a GeoPackage that is in WGS 84 but is missing the CRS metadata, you can add it with the following command:

```
gdal vector edit --crs=EPSG:4326 in.gpkg out.gpkg --overwrite
```

10.5.1.4 Further Reading

- For working with projections using the C++ API, see *OGR Coordinate Reference Systems and Coordinate Transformation tutorial*.
- For discussions about WKT and GDAL, see *OGC WKT Coordinate System Issues*.

10.5.2 OGR Coordinate Reference Systems and Coordinate Transformation tutorial

10.5.2.1 OGC WKT Coordinate System Issues

This document is intended to discuss some issues that arise in attempting to use OpenGIS Well Known Text descriptions of coordinate systems. It discusses various vendor implementations and issues between the original "Simple Features" specification (ie. SF-SQL 99-049) and the newer Coordinate Transformation Services (CT) specification (01-009) which defines an extended form of WKT.

WKT Implementations

At this time I am aware of at least the following software packages that use some form of WKT internally, or for interchange of coordinate system descriptions:

- Oracle Spatial (WKT is used internally in MDSYS.WKT, loosely SFSQL based)
- ESRI - The Arc8 system's projection engine uses a roughly simple features compatible description for projections. I believe ESRI provided the WKT definition for the simple features spec.
- Cadcorp - Has the ability to read and write CT 1.0 style WKT. Cadcorp wrote the CT spec.
- OGR/GDAL - reads/writes WKT as its internal coordinate system description format. Attempts to support old and new forms as well as forms from ESRI.
- FME - Includes WKT read/write capabilities built on OGR.
- MapGuide - Uses WKT in the SDP data access API. Roughly SF compliant.
- PostGIS - Keeps WKT in the spatial_ref_sys table, but it is up to clients to translate to PROJ.4 format for actual use. I believe the spatial_ref_sys table is populated using OGR generated translations.

Projection Parameters

The various specs do not list a set of projections, and the parameters associated with them. This leads to various selection of parameter names (and sometimes projection names) from different vendors. I have attempted to maintain a list of WKT bindings for different projections as part of my GeoTIFF Projections List registry. Please try to adhere to the projection names and parameters listed there. That list also tries to relate the projections to the GeoTIFF, EPSG and PROJ.4 formulations where possible.

The one case where it isn't followed by a vendor that I am aware of ESRIs definition of Lambert Conformal Conic. In EPSG there is a 1SP and a 2SP form of this. ESRI merges them, and just have different parameters depending on the type.

One other issue is that the CT specification does explicitly list parameters for the Transverse Mercator, LCC 1SP and LCC 2SP projections; however, it lists standard_parallel1 and standard_parallel2 as parameters for LCC 2SP which conflicts with the existing usage of standard_parallel_1 and standard_parallel_2 and conflicts with examples in the same CT spec. My position is that the table in section 10.x of the CT spec is in error and that the widely used form is correct. Note that the table in the CT spec conflicts with other examples in the same spec.

A third issue is the formulation for Albers. While I have used longitude_of_center and latitude_of_center ESRI uses Central_meridian and latitude_of_origin.

ESRI:

```
PROJECTION["Albers"],
PARAMETER["False_Easting",1000000.0],
PARAMETER["False_Northing",0.0],
PARAMETER["Central_Meridian",-126.0],
PARAMETER["Standard_Parallel_1",50.0],
```

(continues on next page)

(continued from previous page)

```
PARAMETER["Standard_Parallel_2",58.5],
PARAMETER["Latitude_Of_Origin",45.0],
```

OGR:

```
PROJECTION["Albers"],
PARAMETER["standard_parallel_1",50],
PARAMETER["standard_parallel_2",58.5],
PARAMETER["longitude_of_center",-126],
PARAMETER["latitude_of_center",45],
PARAMETER["false_easting",1000000],
PARAMETER["false_northing",0],
```

Datum Names

In Simple Features style WKT, the name associated with a datum is the only way to identify the datum. In CT WKT the datum can also have a TOWGS84 parameter indicating its relationship to WGS84, and an AUTHORITY parameter relating it to EPSG or some other authority space. However, in SF WKT the name itself is the only key.

By convention OGR and Cadcorp have translated the datum names in a particular way from the EPSG database in order to produce comparable names. The rule is to convert all non alphanumeric characters to underscores, then to strip any leading, trailing or repeating underscores. This produces well behaved datum names like "Nouvelle_Triangulation_Francaise".

However, other vendors have done different things. ESRI seems to follow a similar convention but prefixes all datum names with "D_" as well, giving names like "D_WGS_1972". Also they have lots of other differences for reasons that are not clear. For instance for what Cadcorp and OGR call "Nouvelle_Triangulation_Francaise", they call it "D_NTF". Oracle appears to use the raw names without cleanup. So for NTF they use "NTF (Paris meridian)".

The short result of this is that it is almost impossible to recognise and compare datums between different Simple Features implementations, though I have had some success in translating ESRI datum names to match Cadcorp/OGR conventions, with some special casing.

Parameter Ordering

It is worthwhile keeping in mind that the BNF grammars for WKT in the SF specs, and the CT spec imply specific orders for most items. For instance the BNF for the PROJCS item in the CT spec is

```
<projected cs> =
  PROJCS["<name>", <geographic cs>, <projection>, {<parameter>,}* <linear unit> {,<twin␣
↪axes>}{,<authority>}]
```

This clearly states that the PROJECTION keyword follows the GEOGCS, followed by the UNIT, AXIS and AUTHORITY items. Providing them out of order is technically a violation of the spec. On the other hand, WKT consumers are encouraged to be flexible on ordering.

Units of PARAMETERs

The linear PARAMETER values in a PROJCS must be in terms of the linear units for that PROJCS. I think the only linear units are the false easting and northing type values. Thus, in common cases like a state plane zone in feet, the false easting and northing will also be in feet.

The angular PARAMETER values in a PROJCS must be in terms of the angular units of the GEOGCS. If the GEOGCS is in gradians, for instance, then all the projection angles must also be in gradians!

Units of PRIMEM

What units should the prime meridian appear in?

- The CT 1.0 specification (7.3.14 PRIMEM) says *"The units of the must be inferred from the context. If the PRIMEM clause occurs inside a GEOGCS, then the longitude units will match those of the geographic coordinate system."* Note: for a geocentric coordinate system, it says *"If the PRIMEM clause occurs inside a GEOCCS, then the units will be in degrees"*.
- The SF-SQL spec (99-049) does not attempt to address the issue of units of the prime meridian.
- Existing ESRI EPSG translation to WKT uses degrees for prime meridian, even when the GEOGCS is in gradians as shown in their translation of EPSG 4807:

```
GEOGCS["GCS_NTF_Paris",
  DATUM["D_NTF",
    SPHEROID["Clarke_1880_IGN",6378249.2,293.46602]],
  PRIMEM["Paris",2.337229166666667],
  UNIT["Grad",0.015707963267948967]]
```

- OGR implements the same interpretation as ESRI for its OGRSpatialReference class: the PRIMEM longitude is always in degrees. See GDAL Ticket #4524

```
GEOGCS["NTF (Paris)",
    DATUM["Nouvelle_Triangulation_Francaise_Paris",
        SPHEROID["Clarke 1880 (IGN)",6378249.2,293.4660212936269,
            AUTHORITY["EPSG","7011"]],
        TOWGS84[-168,-60,320,0,0,0,0],
        AUTHORITY["EPSG","6807"]],
    PRIMEM["Paris",2.33722917,
        AUTHORITY["EPSG","8903"]],
    UNIT["grad",0.01570796326794897,
        AUTHORITY["EPSG","9105"]],
    AUTHORITY["EPSG","4807"]]
```

- Cadcorp implements according to the CT 1.0 specification as shown in their translation of EPSG 4807:

```
GEOGCS["NTF (Paris)",
  DATUM["Nouvelle_Triangulation_Francaise",
    SPHEROID["Clarke 1880 (IGN)",6378249.2,293.466021293627,
      AUTHORITY["EPSG",7011]],
    TOWGS84[-168,-60,320,0,0,0,0],
    AUTHORITY["EPSG",6275]],
  PRIMEM["Paris",2.5969213,
    AUTHORITY["EPSG",8903]],
  UNIT["grad",0.015707963267949,
    AUTHORITY["EPSG",9105]],
  AXIS["Lat",NORTH],
  AXIS["Long",EAST],
  AUTHORITY["EPSG",4807]]
```

- Oracle Spatial 8.1.7 uses the following definition for what I assume is supposed to be EPSG 4807. Interestingly it does not bother with using gradians, and it appears that the prime meridian is expressed in radians with very low precision!

```
GEOGCS [ "Longitude / Latitude (NTF with Paris prime meridian)",
  DATUM ["NTF (Paris meridian)",
    SPHEROID ["Clarke 1880 (IGN)", 6378249.200000, 293.466021]],
  PRIMEM [ "", 0.000649 ],
  UNIT ["Decimal Degree", 0.01745329251994330]]
```

Sign of TOWGS84 Rotations

Discussion

In EPSG there are two methods of defining the 7 parameter Bursa-Wolf parameters, 9606 (position vector 7-parameter) and 9607 (coordinate frame rotation). The only difference is that the sign of the rotation coefficients is reversed between them.

I (Frank Warmerdam) had somehow convinced myself that the TOWGS84 values in WKT were supposed to be done using the sense in 9606 (position vector 7-parameter) and that if I read a 9607 I would need to switch the rotation signs before putting it into a TOWGS84 chunk in WKT.

However, I see in the WKT dump you (Martin from Cadcorp) sent me you are using the 9607 sense. For instance, this item appears to use 9607 values directly without switching the sign.

```
GEOGCS["DHDN",
   DATUM["Deutsche_Hauptdreiecksnetz",
     SPHEROID["Bessel 1841",6377397.155,299.1528128,AUTHORITY["EPSG","7004"]],
     TOWGS84[582,105,414,-1.04,-0.35,3.08,8.3],
     AUTHORITY["EPSG","6314"]],
   PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],
   UNIT["DMSH",0.0174532925199433,AUTHORITY["EPSG","9108"]],
   AXIS["Lat",NORTH],AXIS["Long",EAST],AUTHORITY["EPSG","4314"]]
```

I read over the TOWGS84[] clause in the 1.0 CT spec, and it just talks about them being the Bursa Wolf transformation parameters (on page 22, 7.3.18). I also scanned through to 12.3.15.2 and 12.3.27 and they are nonspecific as to the handedness of the TOWGS84 rotations.

I am seeking a clarification of whether TOWGS84 matches EPSG 9606 or EPSG 9607. Furthermore, I would like to see any future rev of the spec clarify this, referencing the EPSG method definitions.

Martin wrote back that he was uncertain on the correct signage and that the Adam had programmed the Cadcorp implementation empirically, according to what seemed to work for the test data available.

I am prepared to adhere to the Cadcorp sign usage (as per EPSG 9607) if this can be clarified in the specification.

Current state of OGR implementation

OGR imports from/exports to WKT assumes EPSG 9606 convention (position vector 7-parameter), as proj does.

When importing from EPSG parameters expressed with EPSG 9607, it does the appropriate conversion (negating the sign of the rotation terms).

Longitudes Relative to PRIMEM?

Another related question is whether longitudinal projection parameters (ie. central meridian) are relative to the GEOGCS prime meridian or relative to greenwich. While the simplest approach is to treat all longitudes as relative to Greenwich, I somehow convinced myself at one point that the longitudes were intended to be relative to the prime meridian. However, a review of 7.3.11 (describing PARAMETER) in the CT 1.0 spec provides no support for this

opinion, and an inspection of EPSG 25700 in Cadcorp also suggests that the central meridian is relative to greenwich, not the prime meridian.

```
PROJCS["Makassar (Jakarta) / NEIEZ",
    GEOGCS["Makassar (Jakarta)",
        DATUM["Makassar",
            SPHEROID["Bessel 1841",6377397.155,299.1528128,
                AUTHORITY["EPSG","7004"]],
            TOWGS84[0,0,0,0,0,0,0],
            AUTHORITY["EPSG","6257"]],
        PRIMEM["Jakarta",106.807719444444,
            AUTHORITY["EPSG","8908"]],
        UNIT["DMSH",0.0174532925199433,
            AUTHORITY["EPSG","9108"]],
        AXIS["Lat","NORTH"],
        AXIS["Long","EAST"],
        AUTHORITY["EPSG","4804"]],
    PROJECTION["Mercator_1SP",
        AUTHORITY["EPSG","9804"]],
    PARAMETER["latitude_of_origin",0],
    PARAMETER["central_meridian",110],
    PARAMETER["scale_factor",0.997],
    PARAMETER["false_easting",3900000],
    PARAMETER["false_northing",900000],
    UNIT["metre",1,
        AUTHORITY["EPSG","9001"]],
    AXIS["X","EAST"],
    AXIS["Y","NORTH"],
    AUTHORITY["EPSG","25700"]]
```

Based on this, I am proceeding on the assumption that while parameters are in the units of the GEOGCS they are not relative the GEOGCS prime meridian.

Numerical Precision in WKT

The specification does not address the precision to which values in WKT should be stored. Some implementations, such as Oracles apparently, use rather limited precision for parameters such as Scale Factor making it difficult to compare coordinate system descriptions or even to get comparable numerical results.

The best practice is to preserve the original precision as specified in the source database, such as EPSG where possible. Given that many systems do not track precision, at least it is advisable to produce values with the equivalent of the C "%.16g" format, maintaining 16 digits of precision, capturing most of the precision of a double precision IEEE floating point value.

Other Notes

1. ESRI seems to use Equidistant_Cylindrical for what I know as Equirectangular.

History

- 2018: Even Rouault: make it clear that OGR implements EPSG 9606 convention for TOWGS84.
- 2018: Even Rouault: remove mention about CT 1.0 specification (7.3.14 PRIMEM) having an error, and explicitly mentions that OGR uses degrees for PRIMEM longitude.

- 2018: Even Rouault: add hyperlinks
- 2007 or before: Originally written by Frank Warmerdam.

10.5.2.2 Introduction

The *OGRSpatialReference* and *OGRCoordinateTransformation* classes provide respectively services to represent coordinate reference systems (known as CRS or SRS, such as typically a projected CRS associating a map projection with a geodetic datums) and to transform between them. These services are loosely modeled on the OpenGIS Coordinate Transformations specification, and rely on the Well Known Text (WKT) format (in its various versions: OGC WKT 1, ESRI WKT, WKT2:2015 and WKT2:2018) for describing coordinate systems.

10.5.2.3 References and applicable standards

- PROJ documentation: projection methods and coordinate operations
- Spatial Reference: an extensive database of spatial reference systems in standardized formats like Well-Known Text (WKT) or PROJJSON
- ISO:19111 and WKT standards
- GeoTIFF Projections Transform List: understanding formulations of projections in WKT for GeoTIFF
- EPSG Geodesy web page is also a useful resource

10.5.2.4 Defining a Geographic Coordinate Reference System

CRS are encapsulated in the *OGRSpatialReference* class. There are a number of ways of initializing an OGRSpatialReference object to a valid coordinate reference system. There are two primary kinds of CRS. The first is geographic (positions are measured in long/lat) and the second is projected (such as UTM - positions are measured in meters or feet).

A Geographic CRS contains information on the datum (which implies a spheroid described by a semi-major axis, and inverse flattening), prime meridian (normally Greenwich), and an angular units type which is normally degrees. The following code initializes a geographic CRS on supplying all this information along with a user visible name for the geographic CRS.

```
OGRSpatialReference oSRS;

oSRS.SetGeogCS( "My geographic CRS",
                "World Geodetic System 1984",
                "My WGS84 Spheroid",
                SRS_WGS84_SEMIMAJOR, SRS_WGS84_INVFLATTENING,
                "Greenwich", 0.0,
                "degree", SRS_UA_DEGREE_CONV );
```

> **Note**
>
> The abbreviation CS in *OGRSpatialReference::SetGeogCS()* is not appropriate according to current geodesic terminology, and should be understood as CRS

Of these values, the names "My geographic CRS", "My WGS84 Spheroid", "Greenwich" and "degree" are not keys, but are used for display to the user. However, the datum name "World Geodetic System 1984" is used as a key to identify the datum, and should be set to a known value from the EPSG registry, so that appropriate datum transformations can be done during coordinate operations. The list of valid geodetic datum can be seen in the 3rd column of the geodetic_datum.sql file.

> **Note**
>
> In WKT 1, space characters in datum names are normally replaced by underscore. And WGS_1984 is used as an alias of "World Geodetic System 1984"

The OGRSpatialReference has built in support for a few well known CRS, which include "NAD27", "NAD83", "WGS72" and "WGS84" which can be defined in a single call to *OGRSpatialReference::SetWellKnownGeogCS()*.

```
oSRS.SetWellKnownGeogCS( "WGS84" );
```

> **Note**
>
> The abbreviation CS in SetWellKnownGeogCS() is not appropriate according to current geodesic terminology, and should be understood as CRS

Furthermore, any geographic CRS in the EPSG database can be set by its GCS code number if the EPSG database is available.

```
oSRS.SetWellKnownGeogCS( "EPSG:4326" );
```

For serialization, and transmission of projection definitions to other packages, the OpenGIS Well Known Text format for coordinate systems is used. An OGRSpatialReference can be initialized from WKT, or converted back into WKT. As of GDAL 3.0, the default format for WKT export is still OGC WKT 1.

```
char *pszWKT = NULL;
oSRS.SetWellKnownGeogCS( "WGS84" );
oSRS.exportToWkt( &pszWKT );
printf( "%s\n", pszWKT );
CPLFree(pszWKT);
```

outputs:

```
GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563,
AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",0,
AUTHORITY["EPSG","8901"]],UNIT["degree",0.0174532925199433,
AUTHORITY["EPSG","9122"]],AXIS["Latitude",NORTH],AXIS["Longitude",EAST],
AUTHORITY["EPSG","4326"]]
```

or in more readable form:

```
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0,
        AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.0174532925199433,
        AUTHORITY["EPSG","9122"]],
    AXIS["Latitude",NORTH],
    AXIS["Longitude",EAST],
    AUTHORITY["EPSG","4326"]]
```

Starting with GDAL 3.0, the *OGRSpatialReference::exportToWkt()* method accepts options,

```
char *pszWKT = nullptr;
oSRS.SetWellKnownGeogCS( "WGS84" );
const char* apszOptions[] = { "FORMAT=WKT2_2018", "MULTILINE=YES", nullptr };
oSRS.exportToWkt( &pszWKT, apszOptions );
printf( "%s\n", pszWKT );
CPLFree(pszWKT);
```

```
GEOGCRS["WGS 84",
    DATUM["World Geodetic System 1984",
        ELLIPSOID["WGS 84",6378137,298.257223563,
            LENGTHUNIT["metre",1]]],
    PRIMEM["Greenwich",0,
        ANGLEUNIT["degree",0.0174532925199433]],
    CS[ellipsoidal,2],
        AXIS["geodetic latitude (Lat)",north,
            ORDER[1],
            ANGLEUNIT["degree",0.0174532925199433]],
        AXIS["geodetic longitude (Lon)",east,
            ORDER[2],
            ANGLEUNIT["degree",0.0174532925199433]],
    ID["EPSG",4326]]
```

This method with options is available in C as the *OSRExportToWktEx()* function.

The *OGRSpatialReference::importFromWkt()* method can be used to set an OGRSpatialReference from a WKT CRS definition.

10.5.2.5 CRS and axis order

One "detail" that has been omitted in previous sections is the topic of the order of coordinate axis in a CRS. A Geographic CRS is, according to ISO:19111 modeling, made of two main components: a geodetic datum and a coordinate system. For 2D geographic CRS, the coordinate system axes are the longitude and the latitude, and the values along those axes are expressed generally in degree (ancient French-based CRS may use grad).

The order in which they are specified, that is latitude first, longitude second, or the reverse, is a constant matter of confusion and vary depending on conventions used by geodetic authorities, GIS user, file format and protocol specifications, etc. This is the source of various interoperability issues.

Before GDAL 3.0, the *OGRSpatialReference* class did not honour the axis order mandated by the authority defining a CRS and consequently stripped axis order information from the WKT string when the order was latitude first, longitude second. Coordinate transformations using the OGRCoordinateTransformation class also assumed that geographic coordinates passed or returned by the Transform() method of this class used the longitude, latitude order.

Starting with GDAL 3.0, the axis order mandated by the authority defining a CRS is by default honoured by the OGRCoordinateTransformation class, and always exported in WKT1. Consequently CRS created with the "EPSG:4326" or "WGS84" strings use the latitude first, longitude second axis order.

In order to help migration from code bases still using coordinates with the longitude, latitude order, it is possible to attach a metadata information to a OGRSpatialReference instance, to specify that for the purpose of coordinate transformations, the order of values effectively passed or returned, will be longitude, latitude. For that, the following must be called

```
oSRS.SetAxisMappingStrategy(OAMS_TRADITIONAL_GIS_ORDER);
```

The argument passed to *OGRSpatialReference::SetAxisMappingStrategy()* is the data axis to CRS axis mapping strategy.

- OAMS_TRADITIONAL_GIS_ORDER means that for geographic CRS with lat/long order, the data will still be long/lat ordered. Similarly for a projected CRS with northing/easting order, the data will still be easting/northing ordered.
- OAMS_AUTHORITY_COMPLIANT means that the data axis will be identical to the CRS axis. This is the default value when instantiating OGRSpatialReference.
- OAMS_CUSTOM means that the data axes are customly defined with SetDataAxisToSRSAxisMapping().

What has been discussed in this section for the particular case of Geographic CRS also applies to Projected CRS. While most of them use Easting first, Northing second convention, some defined in the EPSG registry use the reverse convention.

Another way to keep using the Traditional GIS order for some specific well known CRS is to calling to *OGRSpatialReference::SetWellKnownGeogCS()* with "CRS27", "CRS83" or "CRS84" instead of "NAD27", "NAD83" and "WGS84" respectively.

```
oSRS.SetWellKnownGeogCS( "CRS84" );
```

10.5.2.6 Defining a Projected CRS

A projected CRS (such as UTM, Lambert Conformal Conic, etc.) requires and underlying geographic CRS as well as a definition for the projection transform used to translate between linear positions (in meters or feet) and angular long/lat positions. The following code defines a UTM zone 17 projected CRS with an underlying geographic CRS (datum) of WGS84.

```
OGRSpatialReference oSRS;

oSRS.SetProjCS( "UTM 17 (WGS84) in northern hemisphere." );
oSRS.SetWellKnownGeogCS( "WGS84" );
oSRS.SetUTM( 17, TRUE );
```

Calling *OGRSpatialReference::SetProjCS()* sets a user name for the projected CRS and establishes that the system is projected. The *OGRSpatialReference::SetWellKnownGeogCS()* associates a geographic coordinate system, and the *OGRSpatialReference::SetUTM()* call sets detailed projection transformation parameters. At this time the above order is important in order to create a valid definition, but in the future the object will automatically reorder the internal representation as needed to remain valid.

> **Caution**
>
> For now, be careful of the order of steps defining an OGRSpatialReference!

The above definition would give a WKT version that looks something like the following. Note that the UTM 17 was expanded into the details transverse mercator definition of the UTM zone.

```
PROJCS["UTM 17 (WGS84) in northern hemisphere.",
    GEOGCS["WGS 84",
        DATUM["WGS_1984",
            SPHEROID["WGS 84",6378137,298.257223563,
                AUTHORITY["EPSG",7030]],
            TOWGS84[0,0,0,0,0,0,0],
            AUTHORITY["EPSG",6326]],
```

(continues on next page)

(continued from previous page)

```
        PRIMEM["Greenwich",0,AUTHORITY["EPSG",8901]],
        UNIT["DMSH",0.0174532925199433,AUTHORITY["EPSG",9108]],
        AXIS["Lat",NORTH],
        AXIS["Long",EAST],
        AUTHORITY["EPSG",4326]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",0],
    PARAMETER["central_meridian",-81],
    PARAMETER["scale_factor",0.9996],
    PARAMETER["false_easting",500000],
    PARAMETER["false_northing",0]]
```

There are methods for many projection methods including *OGRSpatialReference::SetTM()* (Transverse Mercator), *OGRSpatialReference::SetLCC()* (Lambert Conformal Conic), and *OGRSpatialReference::SetMercator()*.

10.5.2.7 Querying Coordinate Reference System

Once an OGRSpatialReference has been established, various information about it can be queried. It can be established if it is a projected or geographic CRS using the *OGRSpatialReference::IsProjected()* and *OGRSpatialReference::IsGeographic()* methods. The *OGRSpatialReference::GetSemiMajor()*, *OGRSpatialReference::GetSemiMinor()* and *OGRSpatialReference::GetInvFlattening()* methods can be used to get information about the spheroid. The *OGRSpatialReference::GetAttrValue()* method can be used to get the PROJCS, GEOGCS, DATUM, SPHEROID, and PROJECTION names strings. The *OGRSpatialReference::GetProjParm()* method can be used to get the projection parameters. The *OGRSpatialReference::GetLinearUnits()* method can be used to fetch the linear units type, and translation to meters.

Note that the names of the projection method and parameters is the one of WKT 1.

The following code demonstrates use of *OGRSpatialReference::GetAttrValue()* to get the projection, and *OGRSpatialReference::GetProjParm()* to get projection parameters. The GetAttrValue() method searches for the first "value" node associated with the named entry in the WKT text representation. The #define'ed constants for projection parameters (such as SRS_PP_CENTRAL_MERIDIAN) should be used when fetching projection parameter with GetProjParm(). The code for the Set methods of the various projections in ogrspatialreference.cpp can be consulted to find which parameters apply to which projections.

```
const char *pszProjection = poSRS->GetAttrValue("PROJECTION");

if( pszProjection == NULL )
{
    if( poSRS->IsGeographic() )
        sprintf( szProj4+strlen(szProj4), "+proj=longlat " );
    else
        sprintf( szProj4+strlen(szProj4), "unknown " );
}
else if( EQUAL(pszProjection,SRS_PT_CYLINDRICAL_EQUAL_AREA) )
{
    sprintf( szProj4+strlen(szProj4),
    "+proj=cea +lon_0=%.9f +lat_ts=%.9f +x_0=%.3f +y_0=%.3f ",
            poSRS->GetProjParm(SRS_PP_CENTRAL_MERIDIAN,0.0),
            poSRS->GetProjParm(SRS_PP_STANDARD_PARALLEL_1,0.0),
            poSRS->GetProjParm(SRS_PP_FALSE_EASTING,0.0),
            poSRS->GetProjParm(SRS_PP_FALSE_NORTHING,0.0) );
```

(continues on next page)

(continued from previous page)

```
}
...
```

10.5.2.8 Coordinate Transformation

The *OGRCoordinateTransformation* class is used for translating positions between different CRS. New transformation objects are created using *OGRCreateCoordinateTransformation()*, and then the *OGRCoordinateTransformation::Transform()* method can be used to convert points between CRS.

```
OGRSpatialReference oSourceSRS, oTargetSRS;
OGRCoordinateTransformation *poCT;
double x, y;

oSourceSRS.importFromEPSG( atoi(papszArgv[i+1]) );
oTargetSRS.importFromEPSG( atoi(papszArgv[i+2]) );

poCT = OGRCreateCoordinateTransformation( &oSourceSRS,
                                          &oTargetSRS );
x = atof( papszArgv[i+3] );
y = atof( papszArgv[i+4] );

if( poCT == NULL || !poCT->Transform( 1, &x, &y ) )
    printf( "Transformation failed.\n" );
else
{
    printf( "(%f,%f) -> (%f,%f)\n",
            atof( papszArgv[i+3] ),
            atof( papszArgv[i+4] ),
            x, y );
}
```

There are a couple of points at which transformations can fail. First, OGRCreateCoordinateTransformation() may fail, generally because the internals recognize that no transformation between the indicated systems can be established, and will return a NULL pointer.

The OGRCoordinateTransformation::Transform() method itself can also fail. This may be as a delayed result of one of the above problems, or as a result of an operation being numerically undefined for one or more of the passed in points. The Transform() function will return TRUE on success, or FALSE if any of the points fail to transform. The point array is left in an indeterminate state on error.

Though not shown above, the coordinate transformation service can take 3D points, and will adjust elevations for elevation differences in spheroids, and datums. Elevations given on a geographic or projected CRS are assumed to be ellipsoidal heights. When using a compound CRS made of a horizontal CRS (geographic or projected) and a vertical CRS, elevations will be related to a vertical datum (mean sea level, gravity based, etc.).

Starting with GDAL 3.0, a time value (generally as a value in decimal years) can also be specified for time-dependent coordinate operations.

The following example shows how to conveniently create a long/lat coordinate system using the same geographic CRS as a projected coordinate system, and using that to transform between projected coordinates and long/lat. The returned coordinates will be in longitude, latitude order due to the call to SetAxisMappingStrategy(OAMS_TRADITIONAL_GIS_ORDER)

```
OGRSpatialReference    oUTM, *poLongLat;
OGRCoordinateTransformation *poTransform;

oUTM.SetProjCS("UTM 17 / WGS84");
oUTM.SetWellKnownGeogCS( "WGS84" );
oUTM.SetUTM( 17 );

poLongLat = oUTM.CloneGeogCS();
poLongLat->SetAxisMappingStrategy(OAMS_TRADITIONAL_GIS_ORDER);

poTransform = OGRCreateCoordinateTransformation( &oUTM, poLongLat );
if( poTransform == NULL )
{
    ...
}

...

if( !poTransform->Transform( nPoints, x, y, z ) )
...
```

10.5.2.9 Advanced Coordinate Transformation

OGRCreateCoordinateTransformation() under-the-hood may determine several candidate coordinate operations transforming from the source CRS to the target CRS. Those candidate coordinate operations each have their own area of use. When Transform() is invoked, it will determine the most appropriate coordinate operation based on the coordinates of the point to transform and area of use. For example, there are several dozens of possible coordinate operations for the NAD27 to WGS84 transformation.

If a bounding box of the area of interest into which coordinates to transform are located is known, it is possible to specify it to restrict the candidate coordinate operations to consider:

```
OGRCoordinateTransformationOptions options;
options.SetAreaOfInterest(-100,40,-99,41);
poTransform = OGRCreateCoordinateTransformation( &oNAD27, &oWGS84, options );
```

For cases where a particular coordinate operation must be used, it is possible to specify it as as a PROJ string (single step operation or multiple step string starting with +proj=pipeline), a WKT2 string describing a CoordinateOperation, or a urn:ogc:def:coordinateOperation:EPSG::XXXX URN

```
OGRCoordinateTransformationOptions options;

// EPSG:8599, NAD27 to WGS 84 (46), 1.15 m, USA - Indiana
options.SetCoordinateOperation(
    "+proj=pipeline +step +proj=axisswap +order=2,1 "
    "+step +proj=unitconvert +xy_in=deg +xy_out=rad "
    "+step +proj=hgridshift +grids=conus "
    "+step +proj=hgridshift +grids=inhpgn.gsb "
    "+step +proj=unitconvert +xy_in=rad +xy_out=deg +step "
    "+proj=axisswap +order=2,1", false );

// or
// options.SetCoordinateOperation(
```

(continues on next page)

(continued from previous page)

```
//      "urn:ogc:def:coordinateOperation:EPSG::8599", false);

poTransform = OGRCreateCoordinateTransformation( &oNAD27, &oWGS84, options );
```

10.5.2.10 Alternate Interfaces

A C interface to the coordinate system services is defined in ogr_srs_api.h, and Python bindings are available via the osr.py module. Methods are close analogs of the C++ methods but C and Python bindings are missing for some C++ methods.

C bindings

```
typedef void *OGRSpatialReferenceH;
typedef void *OGRCoordinateTransformationH;

OGRSpatialReferenceH OSRNewSpatialReference( const char * );
void    OSRDestroySpatialReference( OGRSpatialReferenceH );

int     OSRReference( OGRSpatialReferenceH );
int     OSRDereference( OGRSpatialReferenceH );

void OSRSetAxisMappingStrategy( OGRSpatialReferenceH,
                                OSRAxisMappingStrategy );

OGRErr  OSRImportFromEPSG( OGRSpatialReferenceH, int );
OGRErr  OSRImportFromWkt( OGRSpatialReferenceH, char ** );
OGRErr  OSRExportToWkt( OGRSpatialReferenceH, char ** );
OGRErr  OSRExportToWktEx( OGRSpatialReferenceH, char **,
                        const char* const* papszOptions );

OGRErr  OSRSetAttrValue( OGRSpatialReferenceH hSRS, const char * pszNodePath,
                        const char * pszNewNodeValue );
const char *OSRGetAttrValue( OGRSpatialReferenceH hSRS,
                            const char * pszName, int iChild);

OGRErr  OSRSetLinearUnits( OGRSpatialReferenceH, const char *, double );
double  OSRGetLinearUnits( OGRSpatialReferenceH, char ** );

int     OSRIsGeographic( OGRSpatialReferenceH );
int     OSRIsProjected( OGRSpatialReferenceH );
int     OSRIsSameGeogCS( OGRSpatialReferenceH, OGRSpatialReferenceH );
int     OSRIsSame( OGRSpatialReferenceH, OGRSpatialReferenceH );

OGRErr  OSRSetProjCS( OGRSpatialReferenceH hSRS, const char * pszName );
OGRErr  OSRSetWellKnownGeogCS( OGRSpatialReferenceH hSRS,
                            const char * pszName );

OGRErr  OSRSetGeogCS( OGRSpatialReferenceH hSRS,
                    const char * pszGeogName,
                    const char * pszDatumName,
                    const char * pszEllipsoidName,
                    double dfSemiMajor, double dfInvFlattening,
```

(continues on next page)

(continued from previous page)

```
                    const char * pszPMName ,
                    double dfPMOffset ,
                    const char * pszUnits,
                    double dfConvertToRadians );

double  OSRGetSemiMajor( OGRSpatialReferenceH, OGRErr * );
double  OSRGetSemiMinor( OGRSpatialReferenceH, OGRErr * );
double  OSRGetInvFlattening( OGRSpatialReferenceH, OGRErr * );

OGRErr  OSRSetAuthority( OGRSpatialReferenceH hSRS,
                         const char * pszTargetKey,
                         const char * pszAuthority,
                         int nCode );
OGRErr  OSRSetProjParm( OGRSpatialReferenceH, const char *, double );
double  OSRGetProjParm( OGRSpatialReferenceH hSRS,
                         const char * pszParamName,
                         double dfDefault,
                         OGRErr * );

OGRErr  OSRSetUTM( OGRSpatialReferenceH hSRS, int nZone, int bNorth );
int     OSRGetUTMZone( OGRSpatialReferenceH hSRS, int *pbNorth );

OGRCoordinateTransformationH
OCTNewCoordinateTransformation( OGRSpatialReferenceH hSourceSRS,
                                OGRSpatialReferenceH hTargetSRS );

void OCTDestroyCoordinateTransformation( OGRCoordinateTransformationH );

int OCTTransform( OGRCoordinateTransformationH hCT,
                int nCount, double *x, double *y, double *z );

OGRCoordinateTransformationOptionsH OCTNewCoordinateTransformationOptions(;

int OCTCoordinateTransformationOptionsSetOperation(
    OGRCoordinateTransformationOptionsH hOptions,
    const char* pszCO, int bReverseCO);

int OCTCoordinateTransformationOptionsSetAreaOfInterest(
    OGRCoordinateTransformationOptionsH hOptions,
    double dfWestLongitudeDeg,
    double dfSouthLatitudeDeg,
    double dfEastLongitudeDeg,
    double dfNorthLatitudeDeg);

void OCTDestroyCoordinateTransformationOptions(OGRCoordinateTransformationOptionsH);

OGRCoordinateTransformationH
OCTNewCoordinateTransformationEx( OGRSpatialReferenceH hSourceSRS,
                                OGRSpatialReferenceH hTargetSRS,
                                OGRCoordinateTransformationOptionsH hOptions );
```

Python bindings

```
class osr.SpatialReference
    def __init__(self,obj=None):
    def SetAxisMappingStrategy( self, strategy ):
    def ImportFromWkt( self, wkt ):
    def ExportToWkt(self, options = None):
    def ImportFromEPSG(self,code):
    def IsGeographic(self):
    def IsProjected(self):
    def GetAttrValue(self, name, child = 0):
    def SetAttrValue(self, name, value):
    def SetWellKnownGeogCS(self, name):
    def SetProjCS(self, name = "unnamed" ):
    def IsSameGeogCS(self, other):
    def IsSame(self, other):
    def SetLinearUnits(self, units_name, to_meters ):
    def SetUTM(self, zone, is_north = 1):

class CoordinateTransformation:
    def __init__(self,source,target):
    def TransformPoint(self, x, y, z = 0):
    def TransformPoints(self, points):
```

10.5.2.11 History and implementation considerations

Before GDAL 3.0, the OGRSpatialReference class was strongly tied to OGC WKT (WKT 1) format specified by Coordinate Transformation Services (CT) specification (01-009), and the way it was interpreted by GDAL, which various caveats detailed in the *OGC WKT Coordinate System Issues* page. The class mostly contained an in-memory tree-like representation of WKT 1 strings. The class used to directly implement import and export to OGC WKT 1, WKT-ESRI and PROJ.4 formats. Reprojection services were only available if GDAL had been build against the PROJ library.

Starting with GDAL 3.0, the PROJ >= 6.0 library has become a required dependency of GDAL. PROJ 6 has built-in support for OGC WKT 1, ESRI WKT, OGC WKT 2:2015 and OGC WKT 2:2018 representations. PROJ 6 also implements a C++ object class hierarchy of the ISO-19111 / OGC Abstract Topic 2 "Referencing by coordinate" standard. Consequently the OGRSpatialReference class has been modified to act mostly as a wrapper on top of PROJ PJ* CRS objects, and tries to abstract away from the OGC WKT 1 representation as much as possible. However, for backward compatibility, some methods still expect arguments or return values that are specific of OGC WKT 1. The design of the OGRSpatialReference class is also still monolithic. Users wanting direct and fine grained access to CRS representations might want to directly use the PROJ 6 C or C++ API.

10.6 Workshops

- Raster and vector processing with GDAL, FOSS4G-Europe 2015, Even Rouault

10.7 External tutorials

10.7.1 A Gentle Introduction to GDAL, by Robert Simmon

1. A Gentle Introduction to GDAL
2. Map Projections & gdalwarp

3. Geodesy & Local Map Projections
4. Working with Satellite Data
5. Shaded Relief
6. Visualizing Data
7. Transforming Data
8. Reading Scientific Data Formats

CHAPTER

ELEVEN

COMMUNITY

GDAL's community interacts through *Mailing List*, *GitHub*, and *Chat*. Please feel welcome to ask questions and participate in all of the venues. The *Mailing List* communication channel is for general questions, development discussion, and feedback. The *GitHub* communication channel is for development activities, bug reports, and testing. The *Chat* room is for real-time chat activities such as meetings and interactive debugging sessions.

11.1 Code of Conduct

11.1.1 Code of Conduct

This code of conduct governs how we behave in any GDAL forum or event and whenever we will be judged by our actions. We expect it to be honored by everyone who participates in the GDAL community formally or informally, or claims any affiliation with the project.

11.1.1.1 Introduction

This code of conduct governs how we behave in any GDAL forum or event and whenever we will be judged by our actions. We expect it to be honored by everyone who participates in the GDAL community formally or informally, or claims any affiliation with the GDAL project.

It applies to in-person events (such as conferences and related social events), IRC, public and private mailing lists, the issue tracker, the wiki, blogs, social media, and any other forums which the community uses for communication and interactions.

This code is not exhaustive or complete. It serves to distill our common understanding of a collaborative, shared environment and goals. We expect it to be followed in spirit as much as in the letter, so that it can enrich all of us and the technical communities in which we participate.

11.1.1.2 Diversity Statement

GDAL welcomes and encourages participation by everyone. We are committed to being a community that everyone feels good about joining, and we will always work to treat everyone well. No matter how you identify yourself or how others perceive you: we welcome you. Specific Guidelines

We strive to:

- Be open.

  We invite anyone to participate in our community. We preferably use public methods of communication for project-related messages, unless discussing something sensitive. This applies to messages for help or project-related support, too; not only is a public support request much more likely to result in an answer to a question, it also makes sure that any inadvertent mistakes made by people answering will be more easily detected and corrected.

- Be empathetic, welcoming, friendly, and patient.

  We work together to resolve conflict, assume good intentions, and do our best to act in an empathetic fashion. We may all experience some frustration from time to time, but we do not allow frustration to turn into a personal attack. A community where people feel uncomfortable or threatened is not a productive one. Note that we have a multi-cultural, multi-lingual community and some of us are non-native speakers. We should be respectful when dealing with other community members as well as with people outside our community.

- Be collaborative.

  Our work will be used by other people, and in turn we will depend on the work of others. When we make something for the benefit of GDAL, we are willing to explain to others how it works, so that they can build on the work to make it even better. Any decision we make will affect users and colleagues, and we take those consequences seriously when making decisions.

- Be inquisitive.

  Nobody knows everything! Asking questions early avoids many problems later, so questions are encouraged, though they may be directed to the appropriate forum. Those who are asked should be responsive and helpful, within the context of our shared goal of improving GDAL.

- Be careful in the words that we choose.

  Whether we are participating as professionals or volunteers, we value professionalism in all interactions, and take responsibility for our own speech. Be kind to others. Do not insult or put down other participants.

- Be concise.

  Keep in mind that what you write once will be read by hundreds of persons. Writing a short email means people can understand the conversation as efficiently as possible. Short emails should always strive to be empathetic, welcoming, friendly and patient. When a long explanation is necessary, consider adding a summary.

  Try to bring new ideas to a conversation so that each mail adds something unique to the thread, keeping in mind that the rest of the thread still contains the other messages with arguments that have already been made.

  Try to stay on topic, especially in discussions that are already fairly large.

- Step down considerately.

  Members of every project come and go. When somebody leaves or disengages from the project they should tell people they are leaving and take the proper steps to ensure that others can pick up where they left off. In doing so, they should remain respectful of those who continue to participate in the project and should not misrepresent the project's goals or achievements. Likewise, community members should respect any individual's choice to leave the project.

11.1.1.3 Anti-Harassment

Harassment and other exclusionary behavior are not acceptable. This includes, but is not limited to:

- Personal insults or discriminatory jokes and language, especially those using racist or sexist terms.
- Offensive comments, excessive or unnecessary profanity.
- Intimidation, violent threats or demands.
- Sustained disruption of sessions or events.
- Stalking, harassing photography or recording.
- Unwelcome physical contact or sexual attention.
- Repeated harassment of others. In general, if someone asks you to stop, then stop.
- Posting (or threatening to post) other people's personally identifying information ("doxing").

- Sharing private content, such as emails sent privately or non-publicly, or unlogged forums such as IRC channel history.
- Advocating for, or encouraging, any of the above behavior.

11.1.1.4 Reporting Guidelines

If you believe someone is breaking this code of conduct, you may reply to them, and point to this code of conduct. Such messages may be in public or in private, whatever is most appropriate. Assume good faith; it is more likely that participants are unaware of their bad behavior than that they intentionally try to degrade the quality of the discussion. Should there be difficulties in dealing with the situation, you may report your concerns to event staff, a forum leader or the GDAL Board. Serious or persistent offenders may be expelled from the event or forum by event organizers or forum leaders.

To privately email the code of conduct committee leadership use coc@osgeo.org.

11.1.1.5 Credits

GDAL re-uses the OSGeo Code of Conduct

11.2 AI Tool Use Policy

See *LLM tool policy*.

11.2.1 LLM tool policy

11.2.1.1 Rationale

LLMs are changing software development in many ways. What was once scarce, the time and expertise to *write* software, is now plentiful. This scarcity was a choke point that limited consumption of related resources in open source projects. With that scarcity effectively gone in 2026, the dynamics and economics of a project like GDAL are being disrupted.

A project like GDAL is not simply code. It is also design, review, coordinated refactoring, deprecation scheduling, coordinated communication, distribution, and historical outlook. The plentiful convenience of code writing LLMs do not effectively replace the human maintainers, architects, and documenters doing the jobs that provide much of the value GDAL users actually derive. If they did, LLM contributors would not need to bother to upstream activity to the project at all.

With the explosion of LLM usage in software development, the constrained resource is now "maintenance". It is the time to review your contribution, the time to make it concise, the time to refactor it in to a larger system, and the time to protect the larger software system from disruption, breakage, and performance degradation. Indiscriminate usage of LLMs in open source projects *consume* maintenance, and the GDAL LLM tool policy attempts to conserve that resource.

Additionally, legal systems across the world (including US and EU) have not definitely determined whether LLM outputs are derived works of training data or if LLM-written code can even be copyrighted by a human. This is despite it being latently extracted and originated from open source software in the first place.

11.2.1.2 Policy

Contributors can make **limited use** of LLMs for contributions in GDAL, subject to details mentioned below:

- **Human contributors must be the primary author(s) of GDAL contributions**
- All contributions including code, ticket comments, and commit messages should be fully understood by the author(s) submitting them to the project.
- Submission of vibe-coded contributions is *banned*.

- LLMs may only be used as an improved auto-completion mechanism, or for repeated tasks (mechanical refactoring) that could potentially be completed with a deterministic algorithm.
- Human-coordinated or uncoordinated (OpenClaw, etc) use of agents for submission of contributions to the GDAL repository is *banned*.
- *Any* LLM usage must be indicated by ticket label, comment, or commit message indication and account for what was written by whom/what.
- The contributing human author is ultimately responsible for every line of code, comment, or mailing list interaction they initiate, and all of it is subject to the project's *Code of Conduct*.
- The typical high verbosity of LLM code and text is actively discouraged. More code is more code to maintain. High verbosity contribution (tickets, code, messages, etc) will be seen as indication of LLM-generated content when not labeled otherwise and may be ignored, closed, left unmerged, or removed at maintainers' discretion.

11.2.1.3 Violations

If a maintainer judges that a contribution does not comply with this policy, they should paste the following response to request changes:

```
This PR does not appear to comply with our policy on tool-generated content,
and requires additional justification for why it is valuable enough to the
project for us to review it. Please see our developer policy on
AI-generated contributions:
https://gdal.org/community/ai_tool_policy.html
```

If a contributor fails to rectify their contribution to comply with the policy, maintainers may lock the conversation and/or close the pull request/issue/RFC. In case of repeated violations of our policy, the GDAL project reserves itself the right to temporarily or permanently ban the infringing person/account.

11.2.1.4 Mitigation

The GDAL *Sponsorship Program* is one way your organization can help buffer the cost and disruption of LLMs in keystone projects such as GDAL. The constrained resource is maintenance, not adding more code/capability. The Sponsorship Program financially supports operation of GDAL as an ongoing open source software project, and without it, much of the activity GDAL users take for granted would simply not happen.

11.3 Mailing List

Developers and users of GDAL participate on the GDAL mailing list. It is OK to ask questions about how to use GDAL, how to integrate GDAL into your own software, and report issues that you might have.

http://lists.osgeo.org/mailman/listinfo/gdal-dev

11.4 GitHub

Visit http://github.com/OSGeo/GDAL to file issues you might be having with the software. GitHub is also where you can obtain a current development version of the software in the git revision control system. The GDAL project is eager to take contributions in all forms, and we welcome those who are willing to roll up their sleeves and start filing tickets, pushing code, generating builds, and answering questions.

11.5 Chat

You can find some GDAL developers in the IRC channel #gdal on Libera.Chat. This mechanism is usually reserved for active meetings and other outreach with the community. The *Mailing List* and *GitHub* avenues are going to be more productive communication channels in most situations.

11.6 Social media

You can follow GDAL official news on the following platforms:

- Mastodon: https://mastodon.social/@gdal / `@gdal@mastodon.social`
- LinkedIn: https://www.linkedin.com/company/gdalorg/

11.7 Conferences

11.7.1 Upcoming

- European event:

  FOSS4G Europe 2026, 29 June - 3 July 2026, Timișoara, România

- International event:

  FOSS4G 2026 is the Open Source Geospatial Foundation (OSGeo) annual FOSS4G event.

11.7.2 Previous

GDAL presentations have been given at FOSS4G, the leading annual conference for free and open source geospatial software. It is the event for those interested in GDAL/OGR, other FOSS geospatial technologies and the community around them.

- State of GDAL 3.10 & 3.11 by Even Rouault, Spatialys, at FOSS4G Europe 2025 in Mostar, Bosnia-Herzegovina.
- State of GDAL 3.8 & 3.9 by Even Rouault, Spatialys, at FOSS4G Europe 2024 in Tartu, Estonia. Recording.

- State of GDAL 3.6 & 3.7 by Even Rouault, Spatialys, at FOSS4G 2023 in Prizren, Kosovo. Recording.
- State of GDAL 3.4 & 3.5 by Even Rouault, Spatialys, at FOSS4G 2022 in Firenze, Italy. Recording.
- State of GDAL 3.1, 3.2, 3.3… by Even Rouault, Spatialys, at FOSS4G 2021 in Buenos Aires, Argentina. Recording.
- State of GDAL 2.4 and 3.0 by Even Rouault, Spatialys, at FOSS4G 2019 in Bucharest, Romania. Recording.

11.8 Governance and Community Participation

11.8.1 OSGeo Project Membership

Originally, GDAL has been lead by Frank Warmerdam, the original author of much of GDAL/OGR, though with contributions and input from a variety of people. As of February 2006 GDAL/OGR became a founding project of the Open Source Geospatial Foundation (OSGeo) and began a transition to a more community oriented governance model - in keeping with OSGeo expectations.

Membership as an OSGeo project provides assurances that a variety of best practices are being employed by the GDAL/OGR project, and that users and contributors can be assured of responsible project operation and continuity of the project. In particular:

- A consensus oriented Project Management Committee will be in charge of the project.
- Project source code and contributions will be vetted to ensure code is properly made available, protecting contributors and users of GDAL/OGR.
- Part of the systems infrastructure used by the project is provided by the foundation, with responsible backup and redundancy to minimize disruptions.

11.8.2 Project Steering Committee

As of April 2006 (following the GDAL/OGR 1.3.2 release) the project has been placed in the hands of a Project Steering Committee. This project steering committee operates under the rules of rfc-1, and is overall responsible for decisions related to the GDAL/OGR project. The current members are (@github username):

- Frank Warmerdam (@warmerdam)
- Daniel Morissette (@dmorissette)
- Howard Butler (@hobu)
- Tamas Szekeres (@szekerest)
- Even Rouault (chair) (@rouault)
- Jukka Rahkonen (@jratike80)
- Kurt Schwehr (@schwehr)
- Norman Barker (@normanb)
- Javier Jimenez Shaw (@jjimenezshaw)
- Dan Baston (@dbaston)
- Michael Smith (@msmitherdc)

Past members:

- Andrey Kiselev (@strezen) (retired in 2019)
- Mateusz Łoskot (@mloskot) (retired in 2023)
- Sean Gillies (@sgillies) (retired in 2025)

Note that discussion of proposals to the PSC take place on gdal-dev, and input from all subscribers is welcome. A list of past RFC is available for review.

11.8.3 User survey

Results of the *2025 GDAL user survey* are available for review.

11.8.3.1 2024 GDAL user survey

Results of the 2024 GDAL user survey are available at https://gdal.org.

11.8.3.2 2025 GDAL user survey

Results of GDAL user surveys are available at https://gdal.org.

11.9 GDAL Service Providers

11.9.1 GDAL Service Providers

GDAL is developed and supported by a rich ecosystem of businesses and individuals around the world.

In this page we provide a list of service providers who can assist you in getting the best out of your GDAL investment. Their services can range from training and technical support to help you get started, all the way to specialized development and support services to advance the software and support your organization's mission-critical applications.

> **Note**
>
> This list is non-exhaustive, and there are quite likely other service providers in your region of the world who are not listed here. If you think your organization should be listed here, please submit a pull request to add it to this page.

> **Note**
>
> We take responsibility for the accuracy, reliability or completeness of the provided information for Core Contributors only.

11.9.1.1 Core Contributors

Core Contributor organizations have GDAL Committers and/or PSC members as part of their team and as such they are the closest to the core development of the software and are usually the best to go to for support.

They are close enough to the architecture to know what is possible, and to ensure that any improvement to be made to GDAL gets done in the right way and will be included in future releases of the software.

They wrote the software that you use, so you can trust them to help you get the most out of it.

Spatialys (France) is a company founded by Even Rouault that is dedicated to providing to its clients the best of its expertise around Open Source geospatial software - such as GDAL/OGR, PROJ, MapServer, and QGIS - and Open standards.

Hobu, Inc. (United States) is a sam.gov-registered company founded by Howard Butler with more than twenty years experience supporting the creation, enhancement, and management of geospatial open source geospatial software such as PDAL, GDAL/OGR, PROJ, and open standards such as GeoJSON, PROJJSON, and Cloud Optimized Point Cloud.

11.9.1.2 Contributors

The Contributors listed here usually have power users and experienced integrators as part of their team who are recognized in the GDAL community for their contributions to various aspects of the software over the years and are very well positioned to assist you with GDAL.

None listed yet.

11.9.1.3 Other Service Providers

The service providers in this category offer services around GDAL and related Open Source technologies.

Faunalia (Italy) is a company that operates since more than 15 years in the field of Geographic Information Systems (GIS) based on free and open source software (chiefly QGIS, PostGIS, and more). Our main customers are public administrations, private companies, associations. Faunalia has its main branch in Italy, and operates worldwide. Faunalia provides training services on GDAL.

Kartoza (South Africa) is a South Africa-based Free and Open Source GIS (FOSSGIS) service provider. We use GIS software to solve complex location-related problems for individuals, businesses and governments around the world. Kartoza provides training services on GDAL.

mundialis (Germany) is specialized in the processing of massive geospatial and remote sensing data. Our services centered on the Open Source geospatial systems GRASS GIS, actinia and QGIS are incorporated into our process chains, projects and tailored solutions. We analyse satellite, aerial and drone imagery for land degradation and landuse change. We offer support and training for GRASS GIS and GDAL.

See also

Search for GDAL service providers with the OSGeo Service Providers

11.9.1.4 How to Add a Company

The steps to add a company to the service providers list are outlined in the rfc-79 document.

CHAPTER

TWELVE

SPONSORS

12.1 Gold

12.2 Silver

12.3 Bronze

12.4 Supporter

Dynamic Graphics, Inc.

Kaplan Open Source Consulting

Robin Wilson

Satelligence

T-Kartor

UAV4GEO

12.5 Sponsorship Program

You can support GDAL by direct participation in its development and community, by contracting development via *GDAL Service Providers*, or by financially contributing to the GDAL Sponsorship Program (GSP) as a corporation or individually.

GSP supports development of GDAL activities that no single organization would directly contribute through a service provider or in-kind effort. GSP funds are undirected resources the *Project Steering Committee* manages to repay technical debt, address performance and security issues, and add new project-wide features such as the recently added *Main gdal entry point*.

GDAL collaborates with NumFOCUS, a nonprofit dedicated to supporting the open source scientific computing community, as a NumFOCUS Sponsored Project to have it provide administrative support for the GSP. Donations in support of the GSP are made to NumFOCUS and it provides administrative support of the program for a fee.

> **Note**
>
> NumFOCUS is 501(c)(3) non-profit charity in the United States (and soon in France); as such, donations to NumFOCUS are tax-deductible as allowed by law. As with any donation, you should consult with your personal tax adviser or the IRS about your particular tax situation.

12.5.1 Donations

If your organization benefits from GDAL we recommend joining the group of sponsors above to "pay it forward" and ensure the project has the resources to stay healthy and grow. To learn about the benefits of becoming a sponsor at various levels start with the Sustainable GDAL Sponsorship Prospectus. If you are interested, need help convincing your key decision-makers, or have any questions, don't hesitate to contact `gdal-sponsors@osgeo.org`

12.5.1.1 Levels

GDAL (and NumFOCUS) annual sponsorship levels are as follows:

- Gold – $50,000+ USD
- Silver – $25,000+ USD
- Bronze - $10,000+ USD
- Supporter – $1,000-$10,000 USD

12.5.1.2 Corporate

The GDAL project primarily targets corporate donations due to their efficiency. If you represent an organization the leverages GDAL and would like to coordinate your organization's resources to help sponsor the project, please email `gdal-sponsors@osgeo.org`.

12.5.1.3 Individual

Individuals can target donation to the GDAL project by donating to NumFOCUS through https://numfocus.org/donate-to-gdal.

> **Note**
>
> If your workplace has a corporate giving program, please consider enrolling GDAL in it for a match.

12.5.1.4 Government

Government donations (grants) often have significant constraints on their acquisition, usage, and amount. While the project is happy to accept government support through NumFOCUS, note that it can be difficult to meet all requirements. If you are a government organization that desires to support the project through NumFOCUS, please email `gdal-sponsors@osgeo.org`.

12.5.1.5 T-shirts

Profit from swag orders in the NumFOCUS Spreadshirt online shop directly contribute to the project.

> **Note**
>
> The design comes from Lindsey Nield saying "Every new geospatial tool is just GDAL in a trench coat"

12.5.1.6 Past Sponsors

past_sponsors have been critical to enable many feature, security, and performance improvements to the project over the past three years. The project thanks them for their contribution and hopes they can sponsor the project in the future.

12.5.2 Administration

Howard Butler is the GDAL Sponsorship Program Lead and *Project Steering Committee* member. Contact him for questions or information about the program.

The GDAL Maintainers Meeting is every fourth Thursday at 1300 UTC. Any PSC or PR contributor is welcome to join. Contact Howard for an invite and look for the minutes on the *Mailing List*.

12.5.3 Impact

Since its creation in 2021, the GDAL Sponsorship Program has had a huge impact on the administration and development of the GDAL project. It has allowed the project to work to pay back nearly 30 years of accumulated technical and user experience debt that otherwise would not be addressed. Some specific things the GSP enabled include:

- Conversion to the CMake build system
- Implementing the *Main gdal entry point* CLI https://www.youtube.com/watch?v=ZKdrYm3TiBU
- Security remediation in libgdal and supporting libraries
- The *2024 GDAL user survey* and activities resulting from its direction

> **Note**
>
> You can browse GitHub tickets by searching for the *funded through GSP* label to see a listing of activities made possible by the program. https://github.com/OSGeo/gdal/pulls?q=is%3Apr+label%3A%22funded+through+GSP%22+is%3Aclosed

12.5.4 Related Resources

- Sustainable GDAL Sponsorship Prospectus.
- *Sponsoring frequently asked questions (FAQ)*.
- rfc-80
- rfc-83

12.5.4.1 Sponsoring FAQ

Why does GDAL need sponsorship?

GDAL is the most central piece of Critical Digital Infrastructure in the geospatial world, serving as the primary building block for data management and processing in open source, commercial, and government geospatial software. But most of its resources have gone to the development of new features, while the maintenance burden has only increased with more functionality.

The purpose of sponsor funding is to provide substantial long-acting resources that give the project the ability to address software, testing, and project challenges that do not attract individual sponsorship attention. With a pool of resources that are not earmarked for features, it can attack usability, performance, and modernization challenges that benefit everyone. Sustained funding enables multi-year efforts that do not disrupt the existing GDAL user base, and it will provide the baseline resources to allow day-to-day maintenance operations to continue uninterrupted.

The larger funding levels enabled by sponsorship aim to help GDAL grow to a team of maintainers, as is typical in other leading open source projects. The past few years have seen only one maintainer, who has been stretched thin. The resources provided by this effort allow maintenance activities to continue *and* allow the project to support on-boarding additional developers to tackle various needs of the project that have been ignored due to lack of direct funding interest.

My organization wants to sponsor. How can we do that?

To learn about the benefits of becoming a sponsor at various levels start with the Sustainable GDAL Sponsorship Prospectus. If you are interested, need help convincing your key decision-makers, or have any questions, don't hesitate to contact gdal-sponsors@osgeo.org.

What is NumFOCUS and why is the project using that foundation rather than using OSGeo for this effort?

NumFOCUS is a US-based 501(c)(3) tax-exempt non-profit that is already managing funding for many individual software projects using this model such as Numpy, Jupyter, pandas, Julia, and SciPy. They have staff, policies, procedures, and infrastructure for managing the financial support of open source software projects with this funding model. Many organizations in the initial list of sponsors are already funding projects through NumFOCUS, and adding GDAL to the roster improves efficiency on their side.

OSGeo does not have staff and procedures to manage tracking workloads and payments. OSGeo also does not provide tax-exempt status for contributions. NumFOCUS has established relationships with a large number of the initial sponsors. These properties are why the GDAL PSC has chosen the NumFOCUS path for management of this effort.

So GDAL is a NumFOCUS project now?

Not exactly. GDAL will be a project within NumFOCUS under the "Grantor-Grantee Model". OSGeo is still the primary foundational "home" of GDAL such as it is, but NumFOCUS is providing this financial vehicle and service to the project under the purview of its charter.

What is the project going to do with the money?

- The GDAL PSC will control the purse strings. Contributors seeking resources will submit a proposal to the PSC (RFC-style, but not public) describing the tasks and efforts they will seek to achieve. Any substantial efforts with external impacts will continue to be required to use the GDAL RFC process as described in rfc-1.
- Developers will be able submit requests to the GDAL PSC for 'maintenance work units', which could encompass ticket i/o and related code improvements, CI grooming, mailing list gardening, and fuzzing response activities.
- A significant portion (25% per year if possible) of the resources will be targeted toward *growing* new active developers into the project. Examples of this include soliciting ticket and code contributors with funding if they show interest and aptitude, and providing resources to mentor and support junior developers who are working into roles in the project. These activities are extremely hard to do without financial support.
- Spot resources will be available to attack needs that have difficulty finding funded attention, such as API improvements in support of specific application niches and subsystem refreshes like build, tests, and CI. The GDAL PSC could fund significant RFCs that demonstrate need and agreement on an ad hoc basis.
- The GDAL PSC will delegate some resources to GDAL-related projects and dependencies – libtiff, libgeotiff, PROJ, shapelib, and the various language bindings and libraries. If more resources than the GDAL project itself can use are available for a particular year, the project will open them up to the wider community of related libraries and users to repurpose them.

How can the resources be used?

The funds cannot be used to benefit an individual contributor or be directed by a contributor through the non-profit organization. For example, a fictional Imagery Corp cannot fund NumFOCUS with explicit intent to have those resources used to fix bugs or add features that benefit Imagery Corp. Imagery Corp can still continue to solicit GDAL active contributors, or actively contribute the fixes themselves, to achieve their goals with the software.

How is this going to impact GDAL software releases?

A primary goal of this effort is to provide resources needed to allow the current GDAL maintainer to keep up the existing release schedule and cadence. Without these resources, the schedule was likely to have significantly stretched out to one or two maintenance releases per year.

Can I use some of the funding to support fixing something?

Quite possibly. The GDAL PSC will provide a proposal template where you will need to describe the issue(s), propose the approach and impacts of it, and state your cost to complete the effort.

CHAPTER

THIRTEEN

HOW TO CONTRIBUTE?

There are several ways for users to contribute to GDAL:

- contributing code (bug fixes or enhancements)
- filing high-quality bug reports
- improving documentation
- *financial sponsorship*

CHAPTER

FOURTEEN

THANKS

As of 2024, GDAL has now a 26 year-long existence, and many individuals, companies, and *Sponsors* have contributed to its development and success. While it is always difficult to properly acknowledge all contributions, a few individuals particularly stand out:

- Frank Warmerdam, creator of GDAL, initial chair of its Project Steering Committee (PSC) and maintainer from 1998 to 2012, founding member of OSGeo, and currently Principal Engineer at Planet. GDAL would not be what is is without his outstanding work. Support for GeoTIFF, JPEG, PNG, VRT, Shapefile, CSV, PostGIS and dozens of other popular (or niche) formats, as well as gdal_translate, gdalwarp, ogr2ogr and most other utilities are all to be credited to Frank.
- Howard Butler, director of Hobu Inc., active on GDAL since 2004, contributor to the Python SWIG bindings, PSC member, founding member of OSGeo. Howard was the coordinator of the fundraising that lead to the advent of GDAL 3 and PROJ 6, and since 2021, he has been instrumental in setting up the GDAL Sponsorship Program and serves as its head, liaising with our sponsors and NumFOCUS, and hosting GDAL contributor monthly meetings.
- Jukka Rahkonen, Chief IT Specialist at the National Land Survey of Finland, power-user and long-time passionate of GDAL, PSC member. Over the last 15 years, Jukka has been tirelessly helping users and triaging reports on the issue tracker and other support forums.
- Even Rouault, involved since 2007, main maintainer since 2014, current PSC chair, director of Spatialys. Even has authored dozens of drivers (and happy to have managed to retire a few of them!) and many enhancements. He has also been involved in code modernization, extension of the regression test suite, continuous integration and security efforts.

Among code contributors, we want to acknowledge the key, past or current, documentation or code contributions of the following persons:

- Ben Ahmed Daho Ali: SXF driver
- Jorge Arevalo: PostGIS Raster driver
- Dan Baston: modernization of the Python test suite and Python bindings.
- Norman Barker: JPIPKAK and TileDB drivers, PSC member.
- Dmitry Baryshnikov: SXF and CAD drivers and the GNM component.
- Andrew Bell: gdal_viewshed enhancements.
- Pete Bunting: KEA driver.
- Ragi Yaser Burhum: FileGDB driver.
- Brian Case: LIBKML driver.
- Victor Chernetsky: AmigoCloud driver.
- Benjamin Collins: SWIG Java bindings.

- Christopher Condit: OGR to KML conversion.
- Nyall Dawson: GeoPackage, OpenFileGDB, ODBC/PGeo and OGR core enhancements. Past co-maintainer.
- Jürgen Fischer: NAS driver.
- Alessandro Furieri: Spatialite support in SQLite driver, enhancements in GML driver.
- Andrea Giudiceandrea: documentation fixes
- Ari Jolma: Author of (ex) SWIG Perl bindings.
- Trent Hare: PDS, ISIS2 and ISIS3 drivers.
- Björn Harrtell: FlatGeoBuf driver.
- Thomas Hirsch: SOSI driver.
- François Hissel: Selafin driver.
- Pirmin Kalberer: Interlis drivers.
- Andrey Kiselev: BMP, HDF4, HDF5, L1B, MrSID and RMF support, gdal_grid, past PSC member.
- Dan Jacobson: documentation enhancements.
- Kor de Jong: PCRaster driver.
- Peter A. Jonsson: Docker scripts enhancements.
- Chaitanya Kumar CH: past GDAL maintainer.
- Martin Lambers: GTA driver.
- Martin Landa: VFK driver.
- Kikitte Lee: Two-Arm chains edge tracing algorithm of gdal_polygonize.
- Alexandr Lisovenko: SXF driver
- Mateusz Łoskot: GeoJSON driver, (ex) Windows CE support, past GDAL maintainer.
- Ivan Lucena: Idrisi, Intergraph and Oracle drivers, (ex) Windows build scripts.
- Hugo Mercier: marching square algorithm of gdal_contour
- Idan Miara: gdal-utils Python package.
- Martin Mikita: PDFium backend of the PDF driver.
- Hiroshi Miura: Initial CMake build system.
- Vincent Mora: WAsP driver
- Daniel Morissette: CPL library, MapInfo .mif/.tab and Arc/Info vector coverage (AVC) support, PSC member.
- Denis Nadeau: netCDF support.
- Markus Neteler: various contributions to GDAL documentation
- Adam Nowacki: WMS driver.
- Jens Oberender: KML driver.
- Alessandro Pasotti: co-maintainer since 2022.
- Kai Pastor: contributions to the CMake build system.
- Abel Pau: MiraMon driver.
- Matthew Perry: gdaldem

- Mark Phillips: TIGER driver.
- Lucian Plesea: ESRIC and MRF drivers.
- Petr Pridal: gdal2tiles utility.
- Paul Ramsey: GeoPackage and FileGDB drivers.
- Bas Retsios: MSG driver.
- Didier Richard: Geoconcept driver.
- Kevin Ruland: SWIG bindings refactor.
- Maxim Rylov: Hana driver.
- Michael Scholz: OpenDRIVE driver.
- Kurt Schwehr: code linting through the whole codebase, PSC member.
- Avyav Kumar Singh: enhancements in the OGR geometry support, SFCGAL support.
- Craig de Stigter: port of GDAL Python autotest suite to the pytest framework.
- Michael Sumner: documentation enhancements, vrt:// enhancements.
- Tamas Szekeres: SWIG CSharp bindings, gdal_viewshed, PSC member.
- Alan Thomas: major enhancements in the DXF driver.
- Etienne Tourigny: major enhancements in the netCDF driver.
- Philippe Vachon: COASP, GFF, JaxaPalsar, TSX drivers.
- Stephane Villeneuve: OGR Feature Style support, MapInfo .mif.
- Yorick de Wid: LVBAG driver.
- Chris Yesson: gdaldem

Graphical contributions:

- Marin Byrne: for producing the current GDAL icon set (based on the earlier version by Martin Daly).
- Darek Krawczyk: for producing design of the initial GDAL Team Member t-shirt (based on Marin's and Martin's graphics).
- Joe Morrison: for producing the new GDAL t-shirt

Packaging:

- Alpine Linux: Holger Jaekel
- Debian: Bas Couwenberg, Francesco Paolo Lovergine
- Conda-Forge: Filipe Fernandes, Daryl Herzmann and others
- Fedora: Sandro Mani
- GISInternals: Tamas Szekeres
- Homebrew: GDAL Homebrew maintainers
- MS4W: Jeff McKenna
- OpenBSD: Landry Breuil
- OSGeo4W: Jürgen Fischer
- netBSD: Greg Troxel

- Spack: Adam J. Stewart
- Ubuntu: Angelos Tzotsos
- vcpkg: Kai Pastor

You can also consult Frank Warmerdam's sponsors, acknowledgments and credits for the period 1998-2010.

BIBLIOGRAPHY

[Wang2000] Generating Viewsheds without Using Sightlines. Wang, Jianjun, Robinson, Gary J., and White, Kevin. Photogrammetric Engineering and Remote Sensing. p81. https://www.asprs.org/wp-content/uploads/pers/2000journal/january/2000_jan_87-90.pdf

INDEX

Symbols

-3d
    gdal_contour command line option, 535
    gdal_rasterize command line option, 564
-8
    gdal_polygonize command line option, 560
-A
    command line option, 530
-E
    gdallocationinfo command line option, 615
    gdaltransform command line option, 631
-R
    gdal_vsi_delete command line option, 497
    gdal_vsi_list command line option, 499
-V
    gdalsrsinfo command line option, 620
--3d
    gdal_raster_contour command line option, 126
    gdal_vector_rasterize command line option, 423
--A_band
    command line option, 530
--NoDataValue
    command line option, 530
---format
    gdal_dataset_copy command line option, 493
    gdal_dataset_delete command line option, 494
    gdal_dataset_rename command line option, 494
---jpeg-quality
    gdal2tiles command line option, 527
--abs
    gdal_vsi_list command line option, 499
--absolute-path
    command line option, 513
    gdal_raster_footprint command line option, 144
    gdal_raster_index command line option, 155
    gdal_raster_mosaic command line option, 161
    gdal_raster_stack command line option, 252
    gdal_vector_index command line option, 377
    gdal_vsi_list command line option, 499
--accept-different-crs
    gdal_vector_index command line option, 377
--accept-different-schemas
    gdal_vector_index command line option, 377
--active-geometry
    gdal_vector_buffer command line option, 292
    gdal_vector_concave_hull command line option, 325
    gdal_vector_convex_hull command line option, 333
    gdal_vector_dissolve command line option, 340
    gdal_vector_explode_collections command line option, 365
    gdal_vector_make_valid command line option, 390
    gdal_vector_segmentize command line option, 442
    gdal_vector_set_geom_type command line option, 451
    gdal_vector_simplify command line option, 458
    gdal_vector_swap_xy command line option, 470
--active-layer
    gdal_vector_buffer command line option, 292
    gdal_vector_clean_coverage command line option, 308
    gdal_vector_clip command line option, 312
    gdal_vector_concave_hull command line option, 325

gdal_vector_convex_hull command line option, 333
gdal_vector_dissolve command line option, 340
gdal_vector_edit command line option, 344
gdal_vector_explode_collections command line option, 365
gdal_vector_filter command line option, 348
gdal_vector_make_valid command line option, 390
gdal_vector_reproject command line option, 434
gdal_vector_segmentize command line option, 442
gdal_vector_select command line option, 439
gdal_vector_set_field_type command line option, 446
gdal_vector_set_geom_type command line option, 451
gdal_vector_simplify command line option, 458
gdal_vector_simplify_coverage command line option, 462
gdal_vector_swap_xy command line option, 470
--add
gdal_vector_rasterize command line option, 423
--add-alpha
gdal_raster_clean_collar command line option, 115
gdal_raster_clip command line option, 118
gdal_raster_color_map command line option, 122
gdal_raster_mosaic command line option, 161
gdal_raster_reproject command line option, 224
gdal_raster_tile command line option, 256
--add-extra-fields
gdal_vector_combine command line option, 316
--add-mask
gdal_raster_clean_collar command line option, 115
--algorithm
gdal_raster_clean_collar command line option, 115
gdal_raster_tri command line option, 265
--allBands
command line option, 532
--all-input-field
gdal_vector_layer_algebra command line option, 383
--all-method-field
gdal_vector_layer_algebra command line option, 383
--all-touched
gdal_vector_rasterize command line option, 423
--allow-bbox-outside-source
gdal_raster_clip command line option, 118
--allow-holes
gdal_vector_concave_hull command line option, 325
--altitude
gdal_raster_hillshade command line option, 151
--angle
gdal_vector_grid command line option, 370–373
--append
command line option, 513
gdal_raster_as_features command line option, 93
gdal_raster_aspect command line option, 96
gdal_raster_blend command line option, 99
gdal_raster_calc command line option, 112
gdal_raster_convert command line option, 131
gdal_raster_create command line option, 135
gdal_raster_fill_nodata command line option, 148
gdal_raster_footprint command line option, 145
gdal_raster_hillshade command line option, 152
gdal_raster_index command line option, 156
gdal_raster_mosaic command line option, 162
gdal_raster_neighbors command line option, 167
gdal_raster_pansharpen command line option, 180
gdal_raster_polygonize command line option, 213
gdal_raster_proximity command line option, 216
gdal_raster_reclassify command line option, 221
gdal_raster_reproject command line option, 226
gdal_raster_resize command line option,

230
gdal_raster_roughness command line option, 235
gdal_raster_scale command line option, 239
gdal_raster_select command line option, 241
gdal_raster_sieve command line option, 249
gdal_raster_slope command line option, 247
gdal_raster_stack command line option, 253
gdal_raster_tpi command line option, 262
gdal_raster_tri command line option, 265
gdal_raster_unscale command line option, 267
gdal_raster_viewshed command line option, 275
gdal_raster_write command line option, 278
gdal_raster_zonal_stats command line option, 282
gdal_vector_buffer command line option, 292
gdal_vector_check_coverage command line option, 297
gdal_vector_check_geometry command line option, 302
gdal_vector_clean_coverage command line option, 308
gdal_vector_clip command line option, 312
gdal_vector_combine command line option, 317
gdal_vector_concat command line option, 321
gdal_vector_concave_hull command line option, 325
gdal_vector_convert command line option, 329
gdal_vector_convex_hull command line option, 333
gdal_vector_dissolve command line option, 340
gdal_vector_edit command line option, 344
gdal_vector_explode_collections command line option, 365
gdal_vector_filter command line option, 348
gdal_vector_index command line option, 378
gdal_vector_layer_algebra command line option, 383
gdal_vector_make_point command line option, 386
gdal_vector_make_valid command line option, 390
gdal_vector_partition command line option, 396
gdal_vector_rename_layer command line option, 430
gdal_vector_reproject command line option, 434
gdal_vector_segmentize command line option, 442
gdal_vector_select command line option, 439
gdal_vector_set_field_type command line option, 446
gdal_vector_set_geom_type command line option, 451
gdal_vector_simplify command line option, 458
gdal_vector_simplify_coverage command line option, 462
gdal_vector_sql command line option, 466
gdal_vector_swap_xy command line option, 470
gdal_vector_write command line option, 476
--approx-stats
gdal_raster_edit command line option, 137
gdal_raster_info command line option, 83
--array
gdal_mdim_convert command line option, 483
gdal_mdim_info command line option, 479
gdal_mdim_mosaic command line option, 487
--array-option
gdal_mdim_convert command line option, 483
gdal_mdim_info command line option, 479
--ascii
gdal_vector_rename_layer command line option, 429
--attribute-name
gdal_raster_polygonize command line option, 213
gdal_vector_rasterize command line option, 423
--aux-xml
gdal_raster_tile command line option, 256
--auxiliary
gdal_raster_edit command line option, 138
--azimuth
gdal_raster_hillshade command line option, 151
--band

gdal_raster_as_features command line option, 93
gdal_raster_aspect command line option, 96
gdal_raster_color_map command line option, 122
gdal_raster_contour command line option, 126
gdal_raster_edit command line option, 138
gdal_raster_fill_nodata command line option, 148
gdal_raster_footprint command line option, 144
gdal_raster_hillshade command line option, 151
gdal_raster_mosaic command line option, 161
gdal_raster_neighbors command line option, 165
gdal_raster_polygonize command line option, 213
gdal_raster_proximity command line option, 216
gdal_raster_roughness command line option, 235
gdal_raster_scale command line option, 238
gdal_raster_select command line option, 241
gdal_raster_sieve command line option, 249
gdal_raster_slope command line option, 246
gdal_raster_stack command line option, 252
gdal_raster_tpi command line option, 262
gdal_raster_tri command line option, 265
gdal_raster_viewshed command line option, 273
gdal_raster_zonal_stats command line option, 281
gdal_vector_rasterize command line option, 423
--band-count
command line option, 514
gdal_raster_create command line option, 134
--base-url
gdal_raster_index command line option, 155
--bbox
command line option, 514
gdal_raster_clip command line option, 118
gdal_raster_create command line option, 134
gdal_raster_edit command line option, 138
gdal_raster_mosaic command line option, 161
gdal_raster_overview_refresh command line option, 176
gdal_raster_reproject command line option, 224
gdal_raster_stack command line option, 252
gdal_vector_clip command line option, 311
gdal_vector_filter command line option, 348
gdal_vector_grid command line option, 369
--bbox-crs
gdal_raster_clip command line option, 118
gdal_raster_reproject command line option, 224
gdal_vector_clip command line option, 311
--bingkey
gdal2tiles command line option, 526
--bit-depth
gdal_raster_pansharpen command line option, 179
gdal_raster_rgb_to_palette command line option, 233
--build
command line option, 520
vector_common_options command line option, 655
--burn
gdal_raster_create command line option, 134
gdal_vector_rasterize command line option, 423
--calc
command line option, 530
gdal_raster_calc command line option, 111
--cflags
command line option, 522
--checksum
gdal_raster_info command line option, 83
--chunk-size
gdal_raster_zonal_stats command line option, 281
--classification
command line option, 1022
--co
command line option, 512, 532
gdal_mdim_convert command line option, 483
gdal_mdim_mosaic command line option, 487
gdal_raster_as_features command line option, 93

gdal_raster_aspect command line option, 96
gdal_raster_blend command line option, 99
gdal_raster_calc command line option, 112
gdal_raster_clean_collar command line option, 116
gdal_raster_clip command line option, 119
gdal_raster_color_map command line option, 123
gdal_raster_contour command line option, 126
gdal_raster_convert command line option, 131
gdal_raster_create command line option, 135
gdal_raster_fill_nodata command line option, 148
gdal_raster_footprint command line option, 145
gdal_raster_hillshade command line option, 152
gdal_raster_index command line option, 156
gdal_raster_materialize command line option, 158
gdal_raster_mosaic command line option, 162
gdal_raster_neighbors command line option, 167
gdal_raster_nodata_to_alpha command line option, 169
gdal_raster_overview_add command line option, 171, 173
gdal_raster_pansharpen command line option, 180
gdal_raster_pixel_info command line option, 208
gdal_raster_polygonize command line option, 213
gdal_raster_proximity command line option, 216
gdal_raster_reclassify command line option, 221
gdal_raster_reproject command line option, 226
gdal_raster_resize command line option, 231
gdal_raster_rgb_to_palette command line option, 233
gdal_raster_roughness command line option, 235
gdal_raster_scale command line option, 239
gdal_raster_select command line option, 241
gdal_raster_set_type command line option, 244
gdal_raster_sieve command line option, 249
gdal_raster_slope command line option, 247
gdal_raster_stack command line option, 253
gdal_raster_tile command line option, 260
gdal_raster_tpi command line option, 262
gdal_raster_tri command line option, 265
gdal_raster_unscale command line option, 267
gdal_raster_viewshed command line option, 275
gdal_raster_write command line option, 277
gdal_raster_zonal_stats command line option, 282
gdal_vector_buffer command line option, 292
gdal_vector_check_coverage command line option, 297
gdal_vector_check_geometry command line option, 302
gdal_vector_clean_coverage command line option, 308
gdal_vector_clip command line option, 312
gdal_vector_combine command line option, 317
gdal_vector_concat command line option, 321
gdal_vector_concave_hull command line option, 325
gdal_vector_convert command line option, 329
gdal_vector_convex_hull command line option, 333
gdal_vector_dissolve command line option, 340
gdal_vector_edit command line option, 344
gdal_vector_explode_collections command line option, 365
gdal_vector_filter command line option, 348
gdal_vector_grid command line option, 368
gdal_vector_index command line option, 378
gdal_vector_layer_algebra command line option, 383
gdal_vector_make_point command line option, 386
gdal_vector_make_valid command line

option, 390
gdal_vector_materialize command line option, 393
gdal_vector_partition command line option, 396
gdal_vector_rasterize command line option, 425
gdal_vector_rename_layer command line option, 430
gdal_vector_reproject command line option, 434
gdal_vector_segmentize command line option, 442
gdal_vector_select command line option, 439
gdal_vector_set_field_type command line option, 446
gdal_vector_set_geom_type command line option, 451
gdal_vector_simplify command line option, 458
gdal_vector_simplify_coverage command line option, 462
gdal_vector_sql command line option, 466
gdal_vector_swap_xy command line option, 470
gdal_vector_write command line option, 475
--color
gdal_raster_clean_collar command line option, 115
--color-count
gdal_raster_rgb_to_palette command line option, 233
--color-interpretation
command line option, 514
gdal_raster_edit command line option, 139
--color-map
gdal_raster_color_map command line option, 122
gdal_raster_edit command line option, 138
gdal_raster_rgb_to_palette command line option, 233
--color-selection
gdal_raster_color_map command line option, 123
--color-table
command line option, 531
--color-threshold
gdal_raster_clean_collar command line option, 115
--combine-bands
gdal_raster_footprint command line option, 144
--command
gdal_external command line option, 78
--config
command line option, 520
vector_common_options command line option, 656
--connect-diagonal-pixels
gdal_raster_polygonize command line option, 213
gdal_raster_sieve command line option, 249
--content-type
gdal_vsi_sozip command line option, 504
sozip command line option, 691
--contry-code
command line option, 1022
--convention
gdal_driver_zarr_add_georeferencing_convention command line option, 518
gdal_raster_aspect command line option, 96
gdal_raster_tile command line option, 256
--convex-hull
gdal_raster_footprint command line option, 144
--coordinate-system
gdal_raster_footprint command line option, 144
--copy-metadata
gdal_raster_create command line option, 134
--copy-overviews
gdal_raster_create command line option, 134
--copy-src-metadata
gdal_raster_tile command line option, 256
--copyright
gdal2tiles command line option, 526
gdal_raster_tile command line option, 259
--creation-option
command line option, 512, 531
gdal_mdim_convert command line option, 483
gdal_mdim_mosaic command line option, 487
gdal_raster_as_features command line option, 93
gdal_raster_aspect command line option, 96
gdal_raster_blend command line option, 99
gdal_raster_calc command line option, 112
gdal_raster_clean_collar command line option, 116
gdal_raster_clip command line option, 119

gdal_raster_color_map command line option, 123
gdal_raster_contour command line option, 126
gdal_raster_convert command line option, 131
gdal_raster_create command line option, 135
gdal_raster_fill_nodata command line option, 148
gdal_raster_footprint command line option, 145
gdal_raster_hillshade command line option, 152
gdal_raster_index command line option, 156
gdal_raster_materialize command line option, 158
gdal_raster_mosaic command line option, 162
gdal_raster_neighbors command line option, 167
gdal_raster_nodata_to_alpha command line option, 169
gdal_raster_overview_add command line option, 173
gdal_raster_pansharpen command line option, 180
gdal_raster_pixel_info command line option, 208
gdal_raster_polygonize command line option, 213
gdal_raster_proximity command line option, 216
gdal_raster_reclassify command line option, 221
gdal_raster_reproject command line option, 226
gdal_raster_resize command line option, 231
gdal_raster_rgb_to_palette command line option, 233
gdal_raster_roughness command line option, 235
gdal_raster_scale command line option, 239
gdal_raster_select command line option, 241
gdal_raster_set_type command line option, 244
gdal_raster_sieve command line option, 249
gdal_raster_slope command line option, 247
gdal_raster_stack command line option, 253
gdal_raster_tile command line option, 260
gdal_raster_tpi command line option, 262
gdal_raster_tri command line option, 265
gdal_raster_unscale command line option, 267
gdal_raster_viewshed command line option, 275
gdal_raster_write command line option, 277
gdal_raster_zonal_stats command line option, 282
gdal_vector_buffer command line option, 292
gdal_vector_check_coverage command line option, 297
gdal_vector_check_geometry command line option, 302
gdal_vector_clean_coverage command line option, 308
gdal_vector_clip command line option, 312
gdal_vector_combine command line option, 317
gdal_vector_concat command line option, 321
gdal_vector_concave_hull command line option, 325
gdal_vector_convert command line option, 329
gdal_vector_convex_hull command line option, 333
gdal_vector_dissolve command line option, 340
gdal_vector_edit command line option, 344
gdal_vector_explode_collections command line option, 365
gdal_vector_filter command line option, 348
gdal_vector_grid command line option, 368
gdal_vector_index command line option, 378
gdal_vector_layer_algebra command line option, 383
gdal_vector_make_point command line option, 386
gdal_vector_make_valid command line option, 390
gdal_vector_materialize command line option, 393
gdal_vector_partition command line option, 396
gdal_vector_rasterize command line option, 425

gdal_vector_rename_layer command line option, 430
gdal_vector_reproject command line option, 434
gdal_vector_segmentize command line option, 442
gdal_vector_select command line option, 439
gdal_vector_set_field_type command line option, 446
gdal_vector_set_geom_type command line option, 451
gdal_vector_simplify command line option, 458
gdal_vector_simplify_coverage command line option, 462
gdal_vector_sql command line option, 466
gdal_vector_swap_xy command line option, 470
gdal_vector_write command line option, 475
rgb2pct command line option, 648

--crs
gdal_raster_create command line option, 134
gdal_raster_edit command line option, 138
gdal_vector_create command line option, 337
gdal_vector_edit command line option, 344
gdal_vector_grid command line option, 369
gdal_vector_rasterize command line option, 423

--crs-format
gdal_raster_info command line option, 83
gdal_vector_info command line option, 353

--curvature-coefficient
gdal_raster_viewshed command line option, 273

--curve
gdal_vector_set_geom_type command line option, 450

--dataset
gdal_raster_edit command line option, 139
gdal_raster_overview_add command line option, 171
gdal_raster_overview_delete command line option, 175
gdal_raster_overview_refresh command line option, 176

--dataset-name-only
gdal_vector_index command line option, 377

--datatype
command line option, 514
gdal_raster_calc command line option, 113
gdal_raster_create command line option, 135
gdal_raster_neighbors command line option, 167
gdal_raster_nodata_to_alpha command line option, 169
gdal_raster_proximity command line option, 217
gdal_raster_reclassify command line option, 221
gdal_raster_scale command line option, 239
gdal_raster_set_type command line option, 244
gdal_raster_unscale command line option, 268
gdal_vector_grid command line option, 369
gdal_vector_rasterize command line option, 424

--debug
command line option, 520, 532
vector_common_options command line option, 656

--densify-distance
gdal_raster_footprint command line option, 144

--depth
gdal_vsi_list command line option, 499

--destination
gdal_dataset_copy command line option, 493
gdal_dataset_rename command line option, 494

--detailed
gdal_dataset_identify command line option, 490
gdal_mdim_info command line option, 479

--dialect
gdal_raster_calc command line option, 111
gdal_vector_info command line option, 352
gdal_vector_rasterize command line option, 424
gdal_vector_sql command line option, 466

--dim
gdal_vector_set_geom_type command line option, 450

--distance
gdal_vector_buffer command line option, 289

--distance-units
gdal_raster_proximity command line option, 216

--elevation-name

gdal_raster_contour command line option, 126
--enable-sozip
gdal_vsi_sozip command line option, 504, 505
sozip command line option, 690
--end-angle
gdal_raster_viewshed command line option, 273
--endcap-style
gdal_vector_buffer command line option, 289
--error-threshold
gdal_raster_reproject command line option, 224
gdal_raster_update command line option, 269
--et
gdal_raster_reproject command line option, 224
gdal_raster_update command line option, 269
--exclude
gdal2tiles command line option, 524
gdal_raster_select command line option, 241
gdal_vector_select command line option, 438
--excluded-values
gdal2tiles command line option, 525
gdal_raster_tile command line option, 259
--excluded-values-pct-threshold
gdal2tiles command line option, 525
gdal_raster_tile command line option, 259
--exp-base
gdal_raster_contour command line option, 126
--exponent
gdal_raster_scale command line option, 238
--extent
command line option, 531
gdal_vector_grid command line option, 368
gdal_vector_rasterize command line option, 423
--external
gdal_raster_overview_add command line option, 172
gdal_raster_overview_delete command line option, 175
gdal_raster_overview_refresh command line option, 176
--feature-limit
gdal_vector_partition command line option, 395
--feature-only
gdal_vector_set_geom_type command line option, 450
--features
gdal_vector_info command line option, 352
--fetched-metadata
command line option, 514
--fid
gdal_vector_create command line option, 338
gdal_vector_info command line option, 352
--field
gdal_vector_create command line option, 337
gdal_vector_partition command line option, 395
--field-name
gdal_vector_set_field_type command line option, 446
--field-strategy
gdal_vector_concat command line option, 320
--field-type
gdal_vector_set_field_type command line option, 446
--fields
gdal_vector_select command line option, 438
--filename
gdal_dataset_delete command line option, 494
gdal_dataset_identify command line option, 489
gdal_vsi_list command line option, 499
--filename-compatible
gdal_vector_rename_layer command line option, 429
--filename-filter
command line option, 513
gdal_raster_index command line option, 155
gdal_vector_index command line option, 377
--fixed-value
gdal_raster_proximity command line option, 216
--flatten
gdal_raster_calc command line option, 111
--force-kml
gdal2tiles command line option, 526
--force-recursive
gdal_dataset_identify command line option, 490

--format
command line option, 512, 520, 531
gdal_dataset_identify command line option, 490
gdal_external command line option, 78
gdal_mdim_convert command line option, 483
gdal_mdim_mosaic command line option, 487
gdal_raster_as_features command line option, 94
gdal_raster_aspect command line option, 97
gdal_raster_blend command line option, 99
gdal_raster_calc command line option, 113
gdal_raster_clean_collar command line option, 116
gdal_raster_clip command line option, 119
gdal_raster_color_map command line option, 123
gdal_raster_contour command line option, 127
gdal_raster_convert command line option, 132
gdal_raster_create command line option, 135
gdal_raster_fill_nodata command line option, 149
gdal_raster_footprint command line option, 146
gdal_raster_hillshade command line option, 153
gdal_raster_index command line option, 156
gdal_raster_info command line option, 83
gdal_raster_materialize command line option, 158
gdal_raster_mosaic command line option, 163
gdal_raster_neighbors command line option, 167
gdal_raster_nodata_to_alpha command line option, 169
gdal_raster_pansharpen command line option, 180
gdal_raster_pixel_info command line option, 206
gdal_raster_polygonize command line option, 213
gdal_raster_proximity command line option, 217
gdal_raster_reclassify command line option, 221
gdal_raster_reproject command line option, 227
gdal_raster_resize command line option, 231
gdal_raster_rgb_to_palette command line option, 234
gdal_raster_roughness command line option, 236
gdal_raster_scale command line option, 239
gdal_raster_select command line option, 242
gdal_raster_set_type command line option, 244
gdal_raster_sieve command line option, 250
gdal_raster_slope command line option, 247
gdal_raster_stack command line option, 253
gdal_raster_tile command line option, 260
gdal_raster_tpi command line option, 263
gdal_raster_tri command line option, 265
gdal_raster_unscale command line option, 268
gdal_raster_viewshed command line option, 275
gdal_raster_write command line option, 277
gdal_raster_zonal_stats command line option, 283
gdal_vector_buffer command line option, 293
gdal_vector_check_coverage command line option, 298
gdal_vector_check_geometry command line option, 303
gdal_vector_clean_coverage command line option, 308
gdal_vector_clip command line option, 313
gdal_vector_combine command line option, 317
gdal_vector_concat command line option, 322
gdal_vector_concave_hull command line option, 326
gdal_vector_convert command line option, 330
gdal_vector_convex_hull command line option, 334
gdal_vector_dissolve command line option, 341
gdal_vector_edit command line option, 345
gdal_vector_explode_collections command line option, 366
gdal_vector_filter command line option,

349
gdal_vector_grid command line option, 368
gdal_vector_index command line option, 378
gdal_vector_info command line option, 352
gdal_vector_layer_algebra command line option, 384
gdal_vector_make_point command line option, 387
gdal_vector_make_valid command line option, 391
gdal_vector_partition command line option, 397
gdal_vector_rasterize command line option, 425
gdal_vector_rename_layer command line option, 430
gdal_vector_reproject command line option, 434
gdal_vector_segmentize command line option, 443
gdal_vector_select command line option, 439
gdal_vector_set_field_type command line option, 447
gdal_vector_set_geom_type command line option, 452
gdal_vector_simplify command line option, 458
gdal_vector_simplify_coverage command line option, 463
gdal_vector_sql command line option, 467
gdal_vector_swap_xy command line option, 471
gdal_vector_write command line option, 475
gdal_vsi_list command line option, 499
vector_common_options command line option, 656
--formats
command line option, 520, 522
ogrinfo command line option, 659
vector_common_options command line option, 655, 656
--full-check
command line option, 508, 510
--gcp
gdal_raster_edit command line option, 139
--geometry
gdal_raster_clip command line option, 118
gdal_raster_update command line option, 269
gdal_vector_clip command line option, 311
--geometry-crs
gdal_raster_clip command line option, 119
gdal_raster_update command line option, 269
gdal_vector_clip command line option, 312
--geometry-field
gdal_vector_check_coverage command line option, 297
gdal_vector_check_geometry command line option, 302
gdal_vector_create command line option, 337
gdal_vector_sort command line option, 455
--geometry-type
gdal_raster_as_features command line option, 93
gdal_vector_create command line option, 337
gdal_vector_edit command line option, 344
gdal_vector_explode_collections command line option, 365
gdal_vector_layer_algebra command line option, 380
gdal_vector_set_geom_type command line option, 450
--googlekey
gdal2tiles command line option, 526
--gradient-alg
gdal_raster_aspect command line option, 96
gdal_raster_hillshade command line option, 151
gdal_raster_slope command line option, 246
--group
gdal_mdim_convert command line option, 483
--group-by
gdal_vector_combine command line option, 316
--group-transactions
gdal_raster_contour command line option, 126
--grow
sozip command line option, 690
--height
gdal_raster_viewshed command line option, 273
--help
command line option, 530, 541, 543, 558, 596, 619, 679
gdal2tiles command line option, 523, 525
gdal2xyz command line option, 528, 529
gdal_contour command line option, 534
gdal_create command line option, 538

gdal_footprint command line option, 545
gdal_grid command line option, 548
gdal_merge command line option, 555
gdal_polygonize command line option, 560
gdal_proximity command line option, 561
gdal_rasterize command line option, 563
gdal_retile command line option, 567
gdal_translate command line option, 570
gdal_viewshed command line option, 578
gdaladdo command line option, 580
gdalattachpct command line option, 587
gdalbuildvrt command line option, 589
gdalcompare command line option, 592
gdalenhance command line option, 602
gdalinfo command line option, 605
gdallocationinfo command line option, 614
gdalmdiminfo command line option, 649
gdalmdimtranslate command line option, 652
gdalsrsinfo command line option, 620
gdaltindex command line option, 625
gdaltransform command line option, 629
gdalwarp command line option, 633
gnmanalyse command line option, 688
gnmmanage command line option, 686
nearblack command line option, 645
ogr2ogr command line option, 667
ogr_layer_algebra command line option, 684
ogrinfo command line option, 657
ogrmerge command line option, 681
ogrtindex command line option, 677
pct2rgb command line option, 646
rgb2pct command line option, 647
sozip command line option, 690
--help-general
command line option, 520, 530, 541, 543, 558, 596, 619, 679
gdal2tiles command line option, 524
gdal2xyz command line option, 528
gdal_contour command line option, 534
gdal_create command line option, 538
gdal_footprint command line option, 545
gdal_grid command line option, 548
gdal_merge command line option, 555
gdal_polygonize command line option, 560
gdal_proximity command line option, 561
gdal_rasterize command line option, 563
gdal_retile command line option, 567
gdal_translate command line option, 570
gdal_viewshed command line option, 578
gdaladdo command line option, 582
gdalattachpct command line option, 587
gdalbuildvrt command line option, 589
gdalcompare command line option, 592
gdalenhance command line option, 602
gdalinfo command line option, 605
gdallocationinfo command line option, 614
gdalmdiminfo command line option, 649
gdalmdimtranslate command line option, 653
gdalsrsinfo command line option, 620
gdaltindex command line option, 625
gdaltransform command line option, 629
gdalwarp command line option, 633
gnmanalyse command line option, 688
gnmmanage command line option, 687
nearblack command line option, 645
ogr2ogr command line option, 667
ogr_layer_algebra command line option, 684
ogrinfo command line option, 657
ogrmerge command line option, 682
ogrtindex command line option, 677
pct2rgb command line option, 646
rgb2pct command line option, 648
sozip command line option, 690
vector_common_options command line option, 656
--hideNoData
command line option, 530
--hide-nodata
gdal_raster_mosaic command line option, 161
gdal_raster_stack command line option, 252
--high-pitch
gdal_raster_viewshed command line option, 273
--hist
gdal_raster_edit command line option, 139
gdal_raster_info command line option, 83
--id-metadata-item
gdal_raster_index command line option, 155
--id-method
gdal_raster_index command line option, 155
--if
gdal_external command line option, 78
gdal_mdim_convert command line option, 484
gdal_mdim_info command line option, 479
gdal_mdim_mosaic command line option, 487
gdal_raster_as_features command line option, 93
gdal_raster_aspect command line option, 97

gdal_raster_blend command line option, 99
gdal_raster_calc command line option, 112
gdal_raster_clip command line option, 119
gdal_raster_compare command line option, 130
gdal_raster_contour command line option, 127
gdal_raster_convert command line option, 132
gdal_raster_create command line option, 135
gdal_raster_fill_nodata command line option, 148
gdal_raster_footprint command line option, 145
gdal_raster_hillshade command line option, 152
gdal_raster_info command line option, 84
gdal_raster_mosaic command line option, 162
gdal_raster_neighbors command line option, 167
gdal_raster_pansharpen command line option, 180
gdal_raster_pixel_info command line option, 207
gdal_raster_polygonize command line option, 213
gdal_raster_proximity command line option, 216
gdal_raster_read command line option, 218
gdal_raster_reclassify command line option, 221
gdal_raster_reproject command line option, 227
gdal_raster_resize command line option, 231
gdal_raster_roughness command line option, 236
gdal_raster_scale command line option, 239
gdal_raster_select command line option, 242
gdal_raster_sieve command line option, 250
gdal_raster_slope command line option, 247
gdal_raster_stack command line option, 253
gdal_raster_tile command line option, 260
gdal_raster_tpi command line option, 263
gdal_raster_tri command line option, 265
gdal_raster_unscale command line option, 267
gdal_raster_update command line option, 270
gdal_raster_viewshed command line option, 275
gdal_raster_zonal_stats command line option, 283
gdal_vector_buffer command line option, 292
gdal_vector_check_coverage command line option, 297
gdal_vector_check_geometry command line option, 302
gdal_vector_clean_coverage command line option, 308
gdal_vector_clip command line option, 312
gdal_vector_combine command line option, 317
gdal_vector_concat command line option, 321
gdal_vector_concave_hull command line option, 325
gdal_vector_convert command line option, 329
gdal_vector_convex_hull command line option, 333
gdal_vector_dissolve command line option, 340
gdal_vector_edit command line option, 345
gdal_vector_explode_collections command line option, 365
gdal_vector_export_schema command line option, 362
gdal_vector_filter command line option, 348
gdal_vector_grid command line option, 369
gdal_vector_info command line option, 353
gdal_vector_layer_algebra command line option, 384
gdal_vector_make_point command line option, 387
gdal_vector_make_valid command line option, 391
gdal_vector_partition command line option, 396
gdal_vector_rasterize command line option, 425
gdal_vector_read command line option, 427
gdal_vector_rename_layer command line option, 430
gdal_vector_reproject command line option, 434
gdal_vector_segmentize command line option, 443
gdal_vector_select command line option,

439
gdal_vector_set_field_type command line option, 447
gdal_vector_set_geom_type command line option, 452
gdal_vector_simplify command line option, 458
gdal_vector_simplify_coverage command line option, 462
gdal_vector_sql command line option, 466
gdal_vector_swap_xy command line option, 471
gdal_vector_update command line option, 474
--ignore-missing-fields
gdal_vector_select command line option, 438
--include-field
gdal_raster_pixel_info command line option, 206
gdal_raster_zonal_stats command line option, 282
gdal_vector_check_geometry command line option, 302
--include-geom
gdal_raster_zonal_stats command line option, 282
--include-row-col
gdal_raster_as_features command line option, 93
--include-valid
gdal_vector_check_coverage command line option, 297
gdal_vector_check_geometry command line option, 302
--include-xy
gdal_raster_as_features command line option, 93
--init
gdal_vector_rasterize command line option, 423
--input
gdal_dataset_check command line option, 492
gdal_mdim_mosaic command line option, 487
gdal_raster_blend command line option, 99
gdal_raster_compare command line option, 129
gdal_raster_create command line option, 134
gdal_raster_pansharpen command line option, 179
gdal_raster_tile command line option, 257
gdal_vector_create command line option, 337
gdal_vsi_sozip command line option, 503, 505–507
--input-crs
gdal_raster_reproject command line option, 224
gdal_vector_concat command line option, 320
gdal_vector_reproject command line option, 433
--input-field
gdal_vector_layer_algebra command line option, 383
--input-field-type
gdal_vector_set_field_type command line option, 446
--input-format
gdal_external command line option, 78
gdal_mdim_convert command line option, 484
gdal_mdim_info command line option, 479
gdal_mdim_mosaic command line option, 487
gdal_raster_as_features command line option, 93
gdal_raster_aspect command line option, 97
gdal_raster_blend command line option, 99
gdal_raster_calc command line option, 112
gdal_raster_clip command line option, 119
gdal_raster_compare command line option, 130
gdal_raster_contour command line option, 127
gdal_raster_convert command line option, 132
gdal_raster_create command line option, 135
gdal_raster_fill_nodata command line option, 148
gdal_raster_footprint command line option, 145
gdal_raster_hillshade command line option, 152
gdal_raster_info command line option, 84
gdal_raster_mosaic command line option, 162
gdal_raster_neighbors command line option, 167
gdal_raster_pansharpen command line option, 180
gdal_raster_pixel_info command line option, 207
gdal_raster_polygonize command line option, 213

gdal_raster_proximity command line option, 216
gdal_raster_read command line option, 218
gdal_raster_reclassify command line option, 221
gdal_raster_reproject command line option, 227
gdal_raster_resize command line option, 231
gdal_raster_roughness command line option, 236
gdal_raster_scale command line option, 239
gdal_raster_select command line option, 242
gdal_raster_sieve command line option, 250
gdal_raster_slope command line option, 247
gdal_raster_stack command line option, 253
gdal_raster_tile command line option, 260
gdal_raster_tpi command line option, 263
gdal_raster_tri command line option, 265
gdal_raster_unscale command line option, 267
gdal_raster_update command line option, 270
gdal_raster_viewshed command line option, 275
gdal_raster_zonal_stats command line option, 283
gdal_vector_buffer command line option, 292
gdal_vector_check_coverage command line option, 297
gdal_vector_check_geometry command line option, 302
gdal_vector_clean_coverage command line option, 308
gdal_vector_clip command line option, 312
gdal_vector_combine command line option, 317
gdal_vector_concat command line option, 321
gdal_vector_concave_hull command line option, 325
gdal_vector_convert command line option, 329
gdal_vector_convex_hull command line option, 333
gdal_vector_dissolve command line option, 340
gdal_vector_edit command line option, 345
gdal_vector_explode_collections command line option, 365
gdal_vector_export_schema command line option, 362
gdal_vector_filter command line option, 348
gdal_vector_grid command line option, 369
gdal_vector_info command line option, 353
gdal_vector_layer_algebra command line option, 384
gdal_vector_make_point command line option, 387
gdal_vector_make_valid command line option, 391
gdal_vector_partition command line option, 396
gdal_vector_rasterize command line option, 425
gdal_vector_read command line option, 427
gdal_vector_rename_layer command line option, 430
gdal_vector_reproject command line option, 434
gdal_vector_segmentize command line option, 443
gdal_vector_select command line option, 439
gdal_vector_set_field_type command line option, 447
gdal_vector_set_geom_type command line option, 452
gdal_vector_simplify command line option, 458
gdal_vector_simplify_coverage command line option, 462
gdal_vector_sql command line option, 466
gdal_vector_swap_xy command line option, 471
gdal_vector_update command line option, 474
--input-layer
gdal_raster_pixel_info command line option, 206
gdal_vector_buffer command line option, 289
gdal_vector_check_coverage command line option, 297
gdal_vector_check_geometry command line option, 302
gdal_vector_clean_coverage command line option, 306
gdal_vector_clip command line option, 312
gdal_vector_concat command line option, 322

gdal_vector_concave_hull command line option, 326
gdal_vector_convert command line option, 330
gdal_vector_convex_hull command line option, 333
gdal_vector_create command line option, 337
gdal_vector_edit command line option, 345
gdal_vector_explode_collections command line option, 365
gdal_vector_export_schema command line option, 361
gdal_vector_filter command line option, 349
gdal_vector_info command line option, 352
gdal_vector_layer_algebra command line option, 380
gdal_vector_make_point command line option, 387
gdal_vector_make_valid command line option, 391
gdal_vector_rasterize command line option, 423
gdal_vector_rename_layer command line option, 429
gdal_vector_reproject command line option, 434
gdal_vector_segmentize command line option, 443
gdal_vector_select command line option, 439
gdal_vector_set_field_type command line option, 447
gdal_vector_set_geom_type command line option, 452
gdal_vector_simplify command line option, 458
gdal_vector_simplify_coverage command line option, 462
gdal_vector_swap_xy command line option, 471
gdal_vector_update command line option, 473
--input-max
gdal_raster_scale command line option, 238
--input-min
gdal_raster_scale command line option, 238
--input-nodata
gdal_raster_contour command line option, 126
gdal_raster_footprint command line option, 144
gdal_raster_mosaic command line option, 161
gdal_raster_reproject command line option, 224
gdal_raster_stack command line option, 252
--input-prefix
gdal_vector_layer_algebra command line option, 383
--interval
gdal_raster_contour command line option, 126
--invert
gdal_vector_rasterize command line option, 423
--invisible-value
gdal_raster_viewshed command line option, 274
--join-style
gdal_vector_buffer command line option, 289
--junk-paths
gdal_vsi_sozip command line option, 503
sozip command line option, 690
--keep-lower-dim
gdal_vector_make_valid command line option, 390
--keep-nested
gdal_vector_combine command line option, 316
--kernel
gdal_raster_neighbors command line option, 165
--key
gdal_vector_update command line option, 473
--kml
gdal_raster_tile command line option, 257
--layer
command line option, 513
gdal_raster_pixel_info command line option, 206
gdal_vector_create command line option, 337
gdal_vector_export_schema command line option, 361
gdal_vector_grid command line option, 368
gdal_vector_info command line option, 352
--layer-creation-option
gdal_raster_as_features command line option, 93
gdal_raster_contour command line option, 127

gdal_raster_footprint command line option, 145
gdal_raster_index command line option, 156
gdal_raster_pixel_info command line option, 208
gdal_raster_polygonize command line option, 213
gdal_raster_zonal_stats command line option, 283
gdal_vector_buffer command line option, 293
gdal_vector_check_coverage command line option, 297
gdal_vector_check_geometry command line option, 302
gdal_vector_clean_coverage command line option, 308
gdal_vector_clip command line option, 313
gdal_vector_combine command line option, 317
gdal_vector_concat command line option, 322
gdal_vector_concave_hull command line option, 326
gdal_vector_convert command line option, 330
gdal_vector_convex_hull command line option, 334
gdal_vector_dissolve command line option, 341
gdal_vector_edit command line option, 345
gdal_vector_explode_collections command line option, 366
gdal_vector_filter command line option, 349
gdal_vector_index command line option, 378
gdal_vector_layer_algebra command line option, 384
gdal_vector_make_point command line option, 387
gdal_vector_make_valid command line option, 391
gdal_vector_materialize command line option, 393
gdal_vector_partition command line option, 397
gdal_vector_rename_layer command line option, 430
gdal_vector_reproject command line option, 434
gdal_vector_segmentize command line option, 443
gdal_vector_select command line option, 439
gdal_vector_set_field_type command line option, 447
gdal_vector_set_geom_type command line option, 452
gdal_vector_simplify command line option, 458
gdal_vector_simplify_coverage command line option, 462
gdal_vector_sql command line option, 467
gdal_vector_swap_xy command line option, 471
gdal_vector_write command line option, 475
--layer-metadata
gdal_vector_edit command line option, 344
--layer-name
gdal_vector_grid command line option, 368
--layer-only
gdal_vector_set_geom_type command line option, 451
--lco
gdal_raster_as_features command line option, 93
gdal_raster_contour command line option, 127
gdal_raster_footprint command line option, 145
gdal_raster_index command line option, 156
gdal_raster_pixel_info command line option, 208
gdal_raster_polygonize command line option, 213
gdal_raster_zonal_stats command line option, 283
gdal_vector_buffer command line option, 293
gdal_vector_check_coverage command line option, 297
gdal_vector_check_geometry command line option, 302
gdal_vector_clean_coverage command line option, 308
gdal_vector_clip command line option, 313
gdal_vector_combine command line option, 317
gdal_vector_concat command line option, 322
gdal_vector_concave_hull command line option, 326
gdal_vector_convert command line option, 330

gdal_vector_convex_hull command line option, 334
gdal_vector_dissolve command line option, 341
gdal_vector_edit command line option, 345
gdal_vector_explode_collections command line option, 366
gdal_vector_filter command line option, 349
gdal_vector_index command line option, 378
gdal_vector_layer_algebra command line option, 384
gdal_vector_make_point command line option, 387
gdal_vector_make_valid command line option, 391
gdal_vector_materialize command line option, 393
gdal_vector_partition command line option, 397
gdal_vector_rename_layer command line option, 430
gdal_vector_reproject command line option, 434
gdal_vector_segmentize command line option, 443
gdal_vector_select command line option, 439
gdal_vector_set_field_type command line option, 447
gdal_vector_set_geom_type command line option, 452
gdal_vector_simplify command line option, 458
gdal_vector_simplify_coverage command line option, 462
gdal_vector_sql command line option, 467
gdal_vector_swap_xy command line option, 471
gdal_vector_write command line option, 475

--legacy
gdal2tiles command line option, 524

--levels
gdal_raster_contour command line option, 126
gdal_raster_overview_add command line option, 172
gdal_raster_overview_refresh command line option, 176

--libs
command line option, 522

--like
gdal_raster_clip command line option, 119
gdal_raster_create command line option, 134
gdal_raster_overview_refresh command line option, 176
gdal_raster_reproject command line option, 224
gdal_vector_clip command line option, 312
gdal_vector_create command line option, 337

--like-layer
gdal_raster_clip command line option, 119
gdal_vector_clip command line option, 312

--like-sql
gdal_raster_clip command line option, 119
gdal_vector_clip command line option, 312

--like-where
gdal_raster_clip command line option, 119
gdal_vector_clip command line option, 312

--limit
gdal_mdim_info command line option, 479
gdal_vector_info command line option, 352

--linear
gdal_vector_set_geom_type command line option, 451

--list
sozip command line option, 690

--list-mdd
gdal_raster_info command line option, 83

--location-field
gdal_raster_footprint command line option, 144

--location-name
gdal_raster_index command line option, 155
gdal_vector_index command line option, 377

--long
gdal_vsi_list command line option, 499

--long-listing
gdal_vsi_list command line option, 499

--low-pitch
gdal_raster_viewshed command line option, 274

--lower-case
gdal_vector_rename_layer command line option, 429

--m
gdal_vector_make_point command line option, 386

--mapml-template
gdal2tiles command line option, 526
gdal_raster_tile command line option, 260

--mapping

gdal_raster_reclassify command line option, 221
--mask
command line option, 514
gdal_raster_fill_nodata command line option, 148
gdal_raster_select command line option, 241
gdal_raster_sieve command line option, 249
--max-distance
gdal_raster_fill_nodata command line option, 148
gdal_raster_proximity command line option, 216
gdal_raster_viewshed command line option, 274
--max-file-size
gdal_vector_partition command line option, 395
--max-length
gdal_vector_rename_layer command line option, 430
gdal_vector_segmentize command line option, 442
--max-name
gdal_raster_contour command line option, 126
--max-pixel-size
command line option, 513
gdal_raster_index command line option, 155
--max-points
gdal_raster_footprint command line option, 144
gdal_vector_grid command line option, 370, 371
--max-points-per-quadrant
gdal_vector_grid command line option, 370, 371, 373
--max-x
gdal_raster_tile command line option, 257
--max-y
gdal_raster_tile command line option, 257
--max-zoom
gdal_raster_tile command line option, 257
--maximum-gap-width
gdal_vector_check_coverage command line option, 297
gdal_vector_clean_coverage command line option, 306
--maybe-visible-value
gdal_raster_viewshed command line option, 275
--mdd
gdal_raster_info command line option, 83
--merge-strategy
gdal_vector_clean_coverage command line option, 306
--metadata
command line option, 513
gdal_raster_create command line option, 134
gdal_raster_edit command line option, 140
gdal_raster_index command line option, 155
gdal_raster_tile command line option, 257
gdal_vector_edit command line option, 344
gdal_vector_index command line option, 377
--metadata-domain
gdal_raster_info command line option, 83
--method
gdal_raster_neighbors command line option, 166
gdal_vector_make_valid command line option, 390
gdal_vector_sort command line option, 455
--method-field
gdal_vector_layer_algebra command line option, 383
--method-layer
gdal_vector_layer_algebra command line option, 383
--method-prefix
gdal_vector_layer_algebra command line option, 383
--min-distance
gdal_raster_viewshed command line option, 274
--min-max
gdal_raster_info command line option, 84
--min-name
gdal_raster_contour command line option, 126
--min-pixel-size
command line option, 513
gdal_raster_index command line option, 155
--min-points
gdal_vector_grid command line option, 370, 371, 373
--min-points-per-quadrant
gdal_vector_grid command line option, 370–373
--min-ring-area
gdal_raster_footprint command line option, 144

--min-size
gdal_raster_overview_add command line option, 172
--min-x
gdal_raster_tile command line option, 257
--min-y
gdal_raster_tile command line option, 257
--min-zoom
gdal_raster_tile command line option, 257
--mitre-limit
gdal_vector_buffer command line option, 289
--mm
gdal_raster_info command line option, 84
--mode
gdal_raster_viewshed command line option, 274
gdal_vector_concat command line option, 320
gdal_vector_update command line option, 473
--mpi
gdal2tiles command line option, 524
--multi
gdal_vector_set_geom_type command line option, 451
--nln
command line option, 513
gdal_raster_contour command line option, 126
gdal_raster_index command line option, 156
gdal_raster_polygonize command line option, 213
gdal_vector_index command line option, 378
--no-alpha
gdal_raster_tile command line option, 257
--no-check-crs
gdal_raster_calc command line option, 112
--no-check-extent
gdal_raster_calc command line option, 112
--no-clip
gdal_raster_scale command line option, 239
--no-create-empty-layers
gdal_vector_check_coverage command line option, 298
gdal_vector_check_geometry command line option, 303
gdal_vector_convert command line option, 330
--no-ct
gdal_raster_info command line option, 84
--no-dither
gdal_raster_rgb_to_palette command line option, 233
--no-edges
gdal_raster_aspect command line option, 96
gdal_raster_hillshade command line option, 151
gdal_raster_roughness command line option, 235
gdal_raster_slope command line option, 246
gdal_raster_tpi command line option, 262
gdal_raster_tri command line option, 265
--no-fl
gdal_raster_info command line option, 84
--no-gcp
gdal_raster_info command line option, 84
--no-input-field
gdal_vector_layer_algebra command line option, 383
--no-intersection-ok
gdal_raster_tile command line option, 257
--no-kml
gdal2tiles command line option, 526
--no-location-field
gdal_raster_footprint command line option, 145
--no-mask
gdal_raster_info command line option, 84
--no-md
gdal_raster_info command line option, 84
--no-method-field
gdal_vector_layer_algebra command line option, 383
--no-nodata
gdal_raster_info command line option, 84
--no-paths
gdal_vsi_sozip command line option, 503
--no-update-overviews
gdal_raster_update command line option, 269
--nodata
command line option, 514
gdal_raster_calc command line option, 112
gdal_raster_create command line option, 134
gdal_raster_edit command line option, 140
gdal_raster_neighbors command line option, 166
gdal_raster_nodata_to_alpha command line option, 169
gdal_raster_pansharpen command line option, 179

gdal_raster_proximity command line option, 216
gdal_vector_grid command line option, 370–373
gdal_vector_rasterize command line option, 424
--nodata-values-pct-threshold
gdal2tiles command line option, 525
gdal_raster_tile command line option, 259
--num-threads
gdal_raster_pansharpen command line option, 179
gdal_raster_reproject command line option, 224
gdal_raster_tile command line option, 257
gdal_raster_viewshed command line option, 274
gdal_vsi_sync command line option, 501
--observer-spacing
gdal_raster_viewshed command line option, 274
--of
command line option, 512
gdal_dataset_identify command line option, 490
gdal_external command line option, 78
gdal_mdim_convert command line option, 483
gdal_mdim_mosaic command line option, 487
gdal_raster_as_features command line option, 94
gdal_raster_aspect command line option, 97
gdal_raster_blend command line option, 99
gdal_raster_calc command line option, 113
gdal_raster_clean_collar command line option, 116
gdal_raster_clip command line option, 119
gdal_raster_color_map command line option, 123
gdal_raster_contour command line option, 127
gdal_raster_convert command line option, 132
gdal_raster_create command line option, 135
gdal_raster_fill_nodata command line option, 149
gdal_raster_footprint command line option, 146
gdal_raster_hillshade command line option, 153
gdal_raster_index command line option, 156
gdal_raster_info command line option, 83
gdal_raster_materialize command line option, 158
gdal_raster_mosaic command line option, 163
gdal_raster_neighbors command line option, 167
gdal_raster_nodata_to_alpha command line option, 169
gdal_raster_pansharpen command line option, 180
gdal_raster_pixel_info command line option, 206
gdal_raster_polygonize command line option, 213
gdal_raster_proximity command line option, 217
gdal_raster_reclassify command line option, 221
gdal_raster_reproject command line option, 227
gdal_raster_resize command line option, 231
gdal_raster_rgb_to_palette command line option, 234
gdal_raster_roughness command line option, 236
gdal_raster_scale command line option, 239
gdal_raster_select command line option, 242
gdal_raster_set_type command line option, 244
gdal_raster_sieve command line option, 250
gdal_raster_slope command line option, 247
gdal_raster_stack command line option, 253
gdal_raster_tile command line option, 260
gdal_raster_tpi command line option, 263
gdal_raster_tri command line option, 265
gdal_raster_unscale command line option, 268
gdal_raster_viewshed command line option, 275
gdal_raster_write command line option, 277
gdal_raster_zonal_stats command line option, 283
gdal_vector_buffer command line option, 293
gdal_vector_check_coverage command line option, 298

gdal_vector_check_geometry command line option, 303
gdal_vector_clean_coverage command line option, 308
gdal_vector_clip command line option, 313
gdal_vector_combine command line option, 317
gdal_vector_concat command line option, 322
gdal_vector_concave_hull command line option, 326
gdal_vector_convert command line option, 330
gdal_vector_convex_hull command line option, 334
gdal_vector_dissolve command line option, 341
gdal_vector_edit command line option, 345
gdal_vector_explode_collections command line option, 366
gdal_vector_filter command line option, 349
gdal_vector_grid command line option, 368
gdal_vector_index command line option, 378
gdal_vector_info command line option, 352
gdal_vector_layer_algebra command line option, 384
gdal_vector_make_point command line option, 387
gdal_vector_make_valid command line option, 391
gdal_vector_partition command line option, 397
gdal_vector_rasterize command line option, 425
gdal_vector_rename_layer command line option, 430
gdal_vector_reproject command line option, 434
gdal_vector_segmentize command line option, 443
gdal_vector_select command line option, 439
gdal_vector_set_field_type command line option, 447
gdal_vector_set_geom_type command line option, 452
gdal_vector_simplify command line option, 458
gdal_vector_simplify_coverage command line option, 463
gdal_vector_sql command line option, 467
gdal_vector_swap_xy command line option, 471
gdal_vector_write command line option, 475
gdal_vsi_list command line option, 499
--off
gdal_raster_contour command line option, 126
--offset
gdal_raster_contour command line option, 126
gdal_raster_edit command line option, 139
--ogr-enabled
command line option, 522
--omit-partitioned-field
gdal_vector_partition command line option, 396
--only-bbox
gdal_raster_clip command line option, 119
--oo
gdal_mdim_convert command line option, 484
gdal_mdim_info command line option, 479
gdal_mdim_mosaic command line option, 487
gdal_raster_as_features command line option, 94
gdal_raster_aspect command line option, 97
gdal_raster_blend command line option, 99
gdal_raster_calc command line option, 112
gdal_raster_clip command line option, 120
gdal_raster_compare command line option, 130
gdal_raster_contour command line option, 127
gdal_raster_convert command line option, 132
gdal_raster_create command line option, 135
gdal_raster_edit command line option, 140
gdal_raster_fill_nodata command line option, 148
gdal_raster_footprint command line option, 146
gdal_raster_hillshade command line option, 152
gdal_raster_info command line option, 84
gdal_raster_mosaic command line option, 163
gdal_raster_neighbors command line option, 167
gdal_raster_overview_add command line option, 173
gdal_raster_overview_delete command line option, 175

gdal_raster_overview_refresh command line option, 177
gdal_raster_pansharpen command line option, 180
gdal_raster_pixel_info command line option, 207
gdal_raster_polygonize command line option, 214
gdal_raster_proximity command line option, 217
gdal_raster_read command line option, 218
gdal_raster_reclassify command line option, 221
gdal_raster_reproject command line option, 227
gdal_raster_resize command line option, 231
gdal_raster_roughness command line option, 236
gdal_raster_scale command line option, 239
gdal_raster_select command line option, 242
gdal_raster_sieve command line option, 250
gdal_raster_slope command line option, 247
gdal_raster_stack command line option, 253
gdal_raster_tile command line option, 260
gdal_raster_tpi command line option, 263
gdal_raster_tri command line option, 265
gdal_raster_unscale command line option, 267
gdal_raster_update command line option, 270
gdal_raster_viewshed command line option, 275
gdal_raster_zonal_stats command line option, 283
gdal_vector_buffer command line option, 293
gdal_vector_check_coverage command line option, 298
gdal_vector_check_geometry command line option, 303
gdal_vector_clean_coverage command line option, 308
gdal_vector_clip command line option, 313
gdal_vector_combine command line option, 317
gdal_vector_concat command line option, 322
gdal_vector_concave_hull command line option, 326
gdal_vector_convert command line option, 330
gdal_vector_convex_hull command line option, 334
gdal_vector_create command line option, 338
gdal_vector_dissolve command line option, 341
gdal_vector_edit command line option, 345
gdal_vector_explode_collections command line option, 366
gdal_vector_export_schema command line option, 362
gdal_vector_filter command line option, 349
gdal_vector_grid command line option, 369
gdal_vector_info command line option, 353
gdal_vector_layer_algebra command line option, 384
gdal_vector_make_point command line option, 387
gdal_vector_make_valid command line option, 391
gdal_vector_partition command line option, 397
gdal_vector_rasterize command line option, 425
gdal_vector_read command line option, 427
gdal_vector_rename_layer command line option, 430
gdal_vector_reproject command line option, 434
gdal_vector_segmentize command line option, 443
gdal_vector_select command line option, 439
gdal_vector_set_field_type command line option, 447
gdal_vector_set_geom_type command line option, 452
gdal_vector_simplify command line option, 458
gdal_vector_simplify_coverage command line option, 463
gdal_vector_sql command line option, 467
gdal_vector_swap_xy command line option, 471
gdal_vector_update command line option, 474
--opacity
gdal_raster_blend command line option, 99
--open-option
gdal_mdim_convert command line option,

484
gdal_mdim_info command line option, 479
gdal_mdim_mosaic command line option, 487
gdal_raster_as_features command line option, 94
gdal_raster_aspect command line option, 97
gdal_raster_blend command line option, 99
gdal_raster_calc command line option, 112
gdal_raster_clip command line option, 120
gdal_raster_compare command line option, 130
gdal_raster_contour command line option, 127
gdal_raster_convert command line option, 132
gdal_raster_create command line option, 135
gdal_raster_edit command line option, 140
gdal_raster_fill_nodata command line option, 148
gdal_raster_footprint command line option, 146
gdal_raster_hillshade command line option, 152
gdal_raster_info command line option, 84
gdal_raster_mosaic command line option, 163
gdal_raster_neighbors command line option, 167
gdal_raster_overview_add command line option, 173
gdal_raster_overview_delete command line option, 175
gdal_raster_overview_refresh command line option, 177
gdal_raster_pansharpen command line option, 180
gdal_raster_pixel_info command line option, 207
gdal_raster_polygonize command line option, 214
gdal_raster_proximity command line option, 217
gdal_raster_read command line option, 218
gdal_raster_reclassify command line option, 221
gdal_raster_reproject command line option, 227
gdal_raster_resize command line option, 231
gdal_raster_roughness command line option, 236
gdal_raster_scale command line option, 239
gdal_raster_select command line option, 242
gdal_raster_sieve command line option, 250
gdal_raster_slope command line option, 247
gdal_raster_stack command line option, 253
gdal_raster_tile command line option, 260
gdal_raster_tpi command line option, 263
gdal_raster_tri command line option, 265
gdal_raster_unscale command line option, 267
gdal_raster_update command line option, 270
gdal_raster_viewshed command line option, 275
gdal_raster_zonal_stats command line option, 283
gdal_vector_buffer command line option, 293
gdal_vector_check_coverage command line option, 298
gdal_vector_check_geometry command line option, 303
gdal_vector_clean_coverage command line option, 308
gdal_vector_clip command line option, 313
gdal_vector_combine command line option, 317
gdal_vector_concat command line option, 322
gdal_vector_concave_hull command line option, 326
gdal_vector_convert command line option, 330
gdal_vector_convex_hull command line option, 334
gdal_vector_create command line option, 338
gdal_vector_dissolve command line option, 341
gdal_vector_edit command line option, 345
gdal_vector_explode_collections command line option, 366
gdal_vector_export_schema command line option, 362
gdal_vector_filter command line option, 349
gdal_vector_grid command line option, 369
gdal_vector_info command line option, 353
gdal_vector_layer_algebra command line option, 384

gdal_vector_make_point command line option, 387
gdal_vector_make_valid command line option, 391
gdal_vector_partition command line option, 397
gdal_vector_rasterize command line option, 425
gdal_vector_read command line option, 427
gdal_vector_rename_layer command line option, 430
gdal_vector_reproject command line option, 434
gdal_vector_segmentize command line option, 443
gdal_vector_select command line option, 439
gdal_vector_set_field_type command line option, 447
gdal_vector_set_geom_type command line option, 452
gdal_vector_simplify command line option, 458
gdal_vector_simplify_coverage command line option, 463
gdal_vector_sql command line option, 467
gdal_vector_swap_xy command line option, 471
gdal_vector_update command line option, 474
--operation
gdal_vector_layer_algebra command line option, 381
--operator
gdal_raster_blend command line option, 99
--optfile
command line option, 520
vector_common_options command line option, 656
--optimization
gdal_vector_rasterize command line option, 424
--optimize-from
sozip command line option, 690
--ot
command line option, 514
gdal_raster_calc command line option, 113
gdal_raster_create command line option, 135
gdal_raster_neighbors command line option, 167
gdal_raster_nodata_to_alpha command line option, 169
gdal_raster_proximity command line option, 217
gdal_raster_reclassify command line option, 221
gdal_raster_scale command line option, 239
gdal_raster_set_type command line option, 244
gdal_raster_unscale command line option, 268
gdal_vector_grid command line option, 369
gdal_vector_rasterize command line option, 424
--out-of-range-value
gdal_raster_viewshed command line option, 274
--outfile
command line option, 530
--output
gdal_dataset_identify command line option, 489
gdal_mdim_mosaic command line option, 487
gdal_raster_materialize command line option, 158
gdal_raster_pixel_info command line option, 206
gdal_raster_tile command line option, 257
gdal_vector_export_schema command line option, 362
gdal_vector_materialize command line option, 393
gdal_vector_partition command line option, 396
gdal_vsi_sozip command line option, 503, 505
--output-crs
command line option, 513
gdal_raster_footprint command line option, 145
gdal_raster_index command line option, 155
gdal_raster_reproject command line option, 225
gdal_vector_concat command line option, 320
gdal_vector_index command line option, 377
gdal_vector_make_point command line option, 386
gdal_vector_reproject command line option, 433
--output-data-type
command line option, 514
gdal_raster_calc command line option, 113
gdal_raster_create command line option,

135
gdal_raster_neighbors command line option, 167
gdal_raster_nodata_to_alpha command line option, 169
gdal_raster_proximity command line option, 217
gdal_raster_reclassify command line option, 221
gdal_raster_scale command line option, 239
gdal_raster_set_type command line option, 244
gdal_raster_unscale command line option, 268
gdal_vector_grid command line option, 369
gdal_vector_rasterize command line option, 424
--output-field-type
gdal_vector_set_field_type command line option, 446
--output-format
command line option, 512
gdal_dataset_identify command line option, 490
gdal_external command line option, 78
gdal_mdim_convert command line option, 483
gdal_mdim_mosaic command line option, 487
gdal_raster_as_features command line option, 94
gdal_raster_aspect command line option, 97
gdal_raster_blend command line option, 99
gdal_raster_calc command line option, 113
gdal_raster_clean_collar command line option, 116
gdal_raster_clip command line option, 119
gdal_raster_color_map command line option, 123
gdal_raster_contour command line option, 127
gdal_raster_convert command line option, 132
gdal_raster_create command line option, 135
gdal_raster_fill_nodata command line option, 149
gdal_raster_footprint command line option, 146
gdal_raster_hillshade command line option, 153
gdal_raster_index command line option, 156
gdal_raster_info command line option, 83
gdal_raster_materialize command line option, 158
gdal_raster_mosaic command line option, 163
gdal_raster_neighbors command line option, 167
gdal_raster_nodata_to_alpha command line option, 169
gdal_raster_pansharpen command line option, 180
gdal_raster_pixel_info command line option, 206
gdal_raster_polygonize command line option, 213
gdal_raster_proximity command line option, 217
gdal_raster_reclassify command line option, 221
gdal_raster_reproject command line option, 227
gdal_raster_resize command line option, 231
gdal_raster_rgb_to_palette command line option, 234
gdal_raster_roughness command line option, 236
gdal_raster_scale command line option, 239
gdal_raster_select command line option, 242
gdal_raster_set_type command line option, 244
gdal_raster_sieve command line option, 250
gdal_raster_slope command line option, 247
gdal_raster_stack command line option, 253
gdal_raster_tile command line option, 260
gdal_raster_tpi command line option, 263
gdal_raster_tri command line option, 265
gdal_raster_unscale command line option, 268
gdal_raster_viewshed command line option, 275
gdal_raster_write command line option, 277
gdal_raster_zonal_stats command line option, 283
gdal_vector_buffer command line option, 293
gdal_vector_check_coverage command line option, 298

gdal_vector_check_geometry command line option, 303
gdal_vector_clean_coverage command line option, 308
gdal_vector_clip command line option, 313
gdal_vector_combine command line option, 317
gdal_vector_concat command line option, 322
gdal_vector_concave_hull command line option, 326
gdal_vector_convert command line option, 330
gdal_vector_convex_hull command line option, 334
gdal_vector_dissolve command line option, 341
gdal_vector_edit command line option, 345
gdal_vector_explode_collections command line option, 366
gdal_vector_filter command line option, 349
gdal_vector_grid command line option, 368
gdal_vector_index command line option, 378
gdal_vector_info command line option, 352
gdal_vector_layer_algebra command line option, 384
gdal_vector_make_point command line option, 387
gdal_vector_make_valid command line option, 391
gdal_vector_partition command line option, 397
gdal_vector_rasterize command line option, 425
gdal_vector_rename_layer command line option, 430
gdal_vector_reproject command line option, 434
gdal_vector_segmentize command line option, 443
gdal_vector_select command line option, 439
gdal_vector_set_field_type command line option, 447
gdal_vector_set_geom_type command line option, 452
gdal_vector_simplify command line option, 458
gdal_vector_simplify_coverage command line option, 463
gdal_vector_sql command line option, 467
gdal_vector_swap_xy command line option, 471
gdal_vector_write command line option, 475
gdal_vsi_list command line option, 499
--output-layer
gdal_raster_as_features command line option, 93
gdal_raster_contour command line option, 126
gdal_raster_footprint command line option, 145
gdal_raster_index command line option, 156
gdal_raster_polygonize command line option, 213
gdal_raster_zonal_stats command line option, 283
gdal_vector_buffer command line option, 290
gdal_vector_check_coverage command line option, 297
gdal_vector_check_geometry command line option, 302
gdal_vector_clean_coverage command line option, 306
gdal_vector_clip command line option, 313
gdal_vector_concat command line option, 321
gdal_vector_concave_hull command line option, 326
gdal_vector_convert command line option, 330
gdal_vector_convex_hull command line option, 334
gdal_vector_create command line option, 337
gdal_vector_edit command line option, 345
gdal_vector_explode_collections command line option, 366
gdal_vector_filter command line option, 349
gdal_vector_index command line option, 378
gdal_vector_layer_algebra command line option, 384
gdal_vector_make_point command line option, 387
gdal_vector_make_valid command line option, 391
gdal_vector_rename_layer command line option, 429, 430
gdal_vector_reproject command line option, 434
gdal_vector_segmentize command line

option, 443
gdal_vector_select command line option, 439
gdal_vector_set_field_type command line option, 447
gdal_vector_set_geom_type command line option, 452
gdal_vector_simplify command line option, 459
gdal_vector_simplify_coverage command line option, 463
gdal_vector_sql command line option, 465
gdal_vector_swap_xy command line option, 471
gdal_vector_update command line option, 474
gdal_vector_write command line option, 476
--output-max
gdal_raster_scale command line option, 238
--output-min
gdal_raster_scale command line option, 239
--output-nodata
gdal_raster_mosaic command line option, 161
gdal_raster_reproject command line option, 225
gdal_raster_rgb_to_palette command line option, 233
gdal_raster_stack command line option, 252
gdal_raster_tile command line option, 257
gdal_raster_viewshed command line option, 274
--output-oo
gdal_raster_as_features command line option, 94
gdal_raster_contour command line option, 127
gdal_raster_polygonize command line option, 214
gdal_raster_zonal_stats command line option, 283
gdal_vector_buffer command line option, 293
gdal_vector_check_coverage command line option, 298
gdal_vector_check_geometry command line option, 303
gdal_vector_clean_coverage command line option, 308
gdal_vector_clip command line option, 313
gdal_vector_combine command line option, 317
gdal_vector_concat command line option, 322
gdal_vector_concave_hull command line option, 326
gdal_vector_convert command line option, 330
gdal_vector_convex_hull command line option, 334
gdal_vector_dissolve command line option, 341
gdal_vector_edit command line option, 345
gdal_vector_explode_collections command line option, 366
gdal_vector_filter command line option, 349
gdal_vector_make_point command line option, 387
gdal_vector_make_valid command line option, 391
gdal_vector_rename_layer command line option, 430
gdal_vector_reproject command line option, 435
gdal_vector_segmentize command line option, 443
gdal_vector_select command line option, 440
gdal_vector_set_field_type command line option, 447
gdal_vector_set_geom_type command line option, 452
gdal_vector_simplify command line option, 459
gdal_vector_simplify_coverage command line option, 463
gdal_vector_sql command line option, 467
gdal_vector_swap_xy command line option, 471
gdal_vector_update command line option, 474
gdal_vector_write command line option, 476
--output-open-option
gdal_raster_as_features command line option, 94
gdal_raster_contour command line option, 127
gdal_raster_polygonize command line option, 214
gdal_raster_zonal_stats command line option, 283
gdal_vector_buffer command line option,

293
gdal_vector_check_coverage command line option, 298
gdal_vector_check_geometry command line option, 303
gdal_vector_clean_coverage command line option, 308
gdal_vector_clip command line option, 313
gdal_vector_combine command line option, 317
gdal_vector_concat command line option, 322
gdal_vector_concave_hull command line option, 326
gdal_vector_convert command line option, 330
gdal_vector_convex_hull command line option, 334
gdal_vector_dissolve command line option, 341
gdal_vector_edit command line option, 345
gdal_vector_explode_collections command line option, 366
gdal_vector_filter command line option, 349
gdal_vector_make_point command line option, 387
gdal_vector_make_valid command line option, 391
gdal_vector_rename_layer command line option, 430
gdal_vector_reproject command line option, 435
gdal_vector_segmentize command line option, 443
gdal_vector_select command line option, 440
gdal_vector_set_field_type command line option, 447
gdal_vector_set_geom_type command line option, 452
gdal_vector_simplify command line option, 459
gdal_vector_simplify_coverage command line option, 463
gdal_vector_sql command line option, 467
gdal_vector_swap_xy command line option, 471
gdal_vector_update command line option, 474
gdal_vector_write command line option, 476
--overlay
gdal_raster_blend command line option, 99
--overview
gdal_raster_footprint command line option, 145
gdal_raster_pixel_info command line option, 207
--overview-resampling
gdal_raster_tile command line option, 257
--overview-src
gdal_raster_overview_add command line option, 172
--overwrite
command line option, 513, 532
gdal_dataset_copy command line option, 493
gdal_dataset_rename command line option, 494
gdal_mdim_convert command line option, 483
gdal_mdim_mosaic command line option, 487
gdal_raster_as_features command line option, 94
gdal_raster_aspect command line option, 97
gdal_raster_blend command line option, 100
gdal_raster_calc command line option, 113
gdal_raster_clean_collar command line option, 116
gdal_raster_clip command line option, 120
gdal_raster_color_map command line option, 123
gdal_raster_contour command line option, 127
gdal_raster_convert command line option, 132
gdal_raster_create command line option, 135
gdal_raster_fill_nodata command line option, 149
gdal_raster_footprint command line option, 146
gdal_raster_hillshade command line option, 153
gdal_raster_index command line option, 157
gdal_raster_materialize command line option, 158
gdal_raster_mosaic command line option, 163
gdal_raster_neighbors command line option, 167
gdal_raster_pansharpen command line option, 180
gdal_raster_pixel_info command line

option, 208
gdal_raster_polygonize command line option, 214
gdal_raster_proximity command line option, 217
gdal_raster_reclassify command line option, 221
gdal_raster_reproject command line option, 227
gdal_raster_resize command line option, 231
gdal_raster_rgb_to_palette command line option, 234
gdal_raster_roughness command line option, 236
gdal_raster_scale command line option, 239
gdal_raster_select command line option, 242
gdal_raster_set_type command line option, 244
gdal_raster_sieve command line option, 250
gdal_raster_slope command line option, 247
gdal_raster_stack command line option, 253
gdal_raster_tpi command line option, 263
gdal_raster_tri command line option, 265
gdal_raster_unscale command line option, 268
gdal_raster_viewshed command line option, 276
gdal_raster_write command line option, 277
gdal_raster_zonal_stats command line option, 283
gdal_vector_buffer command line option, 293
gdal_vector_check_coverage command line option, 298
gdal_vector_check_geometry command line option, 303
gdal_vector_clean_coverage command line option, 308
gdal_vector_clip command line option, 313
gdal_vector_combine command line option, 318
gdal_vector_concat command line option, 322
gdal_vector_concave_hull command line option, 326
gdal_vector_convert command line option, 330
gdal_vector_convex_hull command line option, 334
gdal_vector_dissolve command line option, 341
gdal_vector_edit command line option, 345
gdal_vector_explode_collections command line option, 366
gdal_vector_export_schema command line option, 362
gdal_vector_filter command line option, 349
gdal_vector_grid command line option, 368
gdal_vector_index command line option, 378
gdal_vector_layer_algebra command line option, 384
gdal_vector_make_point command line option, 387
gdal_vector_make_valid command line option, 391
gdal_vector_materialize command line option, 393
gdal_vector_partition command line option, 397
gdal_vector_rasterize command line option, 425
gdal_vector_rename_layer command line option, 431
gdal_vector_reproject command line option, 435
gdal_vector_segmentize command line option, 443
gdal_vector_select command line option, 440
gdal_vector_set_field_type command line option, 447
gdal_vector_set_geom_type command line option, 452
gdal_vector_simplify command line option, 459
gdal_vector_simplify_coverage command line option, 463
gdal_vector_sql command line option, 467
gdal_vector_swap_xy command line option, 471
gdal_vector_write command line option, 476
gdal_vsi_sozip command line option, 503, 505
sozip command line option, 690
--overwrite-layer
command line option, 513
gdal_raster_as_features command line option, 94

gdal_raster_index command line option, 157
gdal_raster_polygonize command line option, 214
gdal_raster_zonal_stats command line option, 283
gdal_vector_buffer command line option, 293
gdal_vector_check_coverage command line option, 298
gdal_vector_check_geometry command line option, 303
gdal_vector_clean_coverage command line option, 309
gdal_vector_clip command line option, 313
gdal_vector_combine command line option, 318
gdal_vector_concat command line option, 322
gdal_vector_concave_hull command line option, 326
gdal_vector_convert command line option, 330
gdal_vector_convex_hull command line option, 334
gdal_vector_dissolve command line option, 341
gdal_vector_edit command line option, 345
gdal_vector_explode_collections command line option, 366
gdal_vector_filter command line option, 349
gdal_vector_index command line option, 378
gdal_vector_layer_algebra command line option, 384
gdal_vector_make_point command line option, 387
gdal_vector_make_valid command line option, 391
gdal_vector_rename_layer command line option, 431
gdal_vector_reproject command line option, 435
gdal_vector_segmentize command line option, 444
gdal_vector_select command line option, 440
gdal_vector_set_field_type command line option, 447
gdal_vector_set_geom_type command line option, 452
gdal_vector_simplify command line option, 459
gdal_vector_simplify_coverage command line option, 463
gdal_vector_sql command line option, 467
gdal_vector_swap_xy command line option, 471
gdal_vector_write command line option, 476
--ovr
gdal_raster_pixel_info command line option, 207
--panchromatic
gdal_raster_pansharpen command line option, 179
--parallel-method
gdal_raster_tile command line option, 257
--partial-refresh-from-projwin
gdaladdo command line option, 583
--partial-refresh-from-source-extent
gdaladdo command line option, 583
--partial-refresh-from-source-timestamp
gdaladdo command line option, 582
--pattern
gdal_vector_partition command line option, 396
--pixel-distance
gdal_raster_clean_collar command line option, 115
--pixel-function
gdal_raster_mosaic command line option, 162
--pixel-function-arg
gdal_raster_mosaic command line option, 162
--pixels
gdal_raster_zonal_stats command line option, 282
--polygonize
gdal_raster_contour command line option, 126
--pos
gdal_raster_pixel_info command line option, 207
gdal_raster_viewshed command line option, 274
--position
gdal_raster_pixel_info command line option, 207
gdal_raster_viewshed command line option, 274
--position-crs
gdal_raster_pixel_info command line option, 207
--position-dataset
gdal_raster_pixel_info command line

option, 206
--power
gdal_vector_grid command line option, 369, 370
--prefix
command line option, 522
--preserve-boundary
gdal_vector_simplify_coverage command line option, 462
--processes
gdal2tiles command line option, 524
--producer-id
command line option, 1022
--producer-name
command line option, 1022
--profile
gdal2tiles command line option, 524
gdal_raster_index command line option, 155
--projectionCheck
command line option, 531
--projwin
command line option, 531
--promote-pixel-value-to-z
gdal_raster_pixel_info command line option, 207
--propagate-nodata
gdal_raster_calc command line option, 112
--quadrant-segments
gdal_vector_buffer command line option, 290
--quiet
command line option, 508, 510, 532, 596, 679
gdal2tiles command line option, 524
gdal_contour command line option, 534
gdal_create command line option, 538
gdal_footprint command line option, 545
gdal_grid command line option, 548
gdal_raster_convert command line option, 132
gdal_raster_reproject command line option, 227
gdal_rasterize command line option, 563
gdal_translate command line option, 570
gdal_vector_convert command line option, 331
gdal_vector_reproject command line option, 435
gdal_vector_sql command line option, 466, 468
gdal_viewshed command line option, 578
gdal_vsi_sozip command line option, 504, 505, 507
gdaladdo command line option, 582
gdalbuildvrt command line option, 589
gdalmdimtranslate command line option, 653
gdaltindex command line option, 625
gdalwarp command line option, 633
nearblack command line option, 645
ogr2ogr command line option, 667
sozip command line option, 690
--radius
gdal_vector_grid command line option, 369–373
--radius1
gdal_vector_grid command line option, 370–373
--radius2
gdal_vector_grid command line option, 370–373
--ratio
gdal_vector_concave_hull command line option, 325
--recurse-paths
sozip command line option, 690
--recursive
command line option, 513
gdal_dataset_identify command line option, 490
gdal_raster_index command line option, 156
gdal_vector_index command line option, 377
gdal_vsi_copy command line option, 496
gdal_vsi_delete command line option, 497
gdal_vsi_list command line option, 499
gdal_vsi_sozip command line option, 503
gdal_vsi_sync command line option, 501
--reference
gdal_raster_compare command line option, 129
--replacement-character
gdal_vector_rename_layer command line option, 429
--report-failures
gdal_dataset_identify command line option, 490
--resampling
gdal2tiles command line option, 524
gdal_raster_overview_add command line option, 172
gdal_raster_overview_refresh command line option, 176
gdal_raster_pansharpen command line option, 179
gdal_raster_pixel_info command line option, 207

gdal_raster_reproject command line option, 225
gdal_raster_resize command line option, 230
gdal_raster_tile command line option, 258
gdal_raster_update command line option, 269
--reserved-characters
gdal_vector_rename_layer command line option, 429
--resolution
command line option, 514
gdal_raster_mosaic command line option, 162
gdal_raster_reproject command line option, 226
gdal_raster_resize command line option, 230
gdal_raster_stack command line option, 252
gdal_vector_grid command line option, 368
gdal_vector_rasterize command line option, 424
--resume
gdal2tiles command line option, 524
gdal_raster_tile command line option, 258
--s_srs
gdal2tiles command line option, 524
--scale
command line option, 1022
gdal_raster_edit command line option, 139
--scale-axes
gdal_mdim_convert command line option, 484
--schema
gdal_vector_create command line option, 337
--scheme
gdal_vector_partition command line option, 396
--sd-filename
gdal_raster_viewshed command line option, 275
--side
gdal_vector_buffer command line option, 290
--simplify-tolerance
gdal_raster_footprint command line option, 145
--single
gdal_vector_set_geom_type command line option, 451
--single-line
gdalsrsinfo command line option, 620
--size
gdal_raster_create command line option, 135
gdal_raster_neighbors command line option, 166
gdal_raster_reproject command line option, 226
gdal_raster_resize command line option, 230
gdal_vector_grid command line option, 368
gdal_vector_rasterize command line option, 424
--size-threshold
gdal_raster_sieve command line option, 249
--skip
gdal_vector_set_geom_type command line option, 451
--skip-all-optional
gdal_raster_compare command line option, 129
--skip-binary
gdal_raster_compare command line option, 129
--skip-blank
gdal_raster_tile command line option, 258
--skip-crs
gdal_raster_compare command line option, 129
--skip-errors
gdal_raster_index command line option, 156
gdal_raster_zonal_stats command line option, 283
gdal_vector_buffer command line option, 293
gdal_vector_check_coverage command line option, 298
gdal_vector_check_geometry command line option, 303
gdal_vector_clean_coverage command line option, 309
gdal_vector_clip command line option, 313
gdal_vector_combine command line option, 318
gdal_vector_concat command line option, 322
gdal_vector_concave_hull command line option, 326
gdal_vector_convert command line option, 330
gdal_vector_convex_hull command line option, 334
gdal_vector_dissolve command line

option, 341
gdal_vector_edit command line option, 345
gdal_vector_explode_collections command line option, 366
gdal_vector_filter command line option, 349
gdal_vector_make_point command line option, 388
gdal_vector_make_valid command line option, 391
gdal_vector_partition command line option, 397
gdal_vector_rename_layer command line option, 431
gdal_vector_reproject command line option, 435
gdal_vector_segmentize command line option, 444
gdal_vector_select command line option, 440
gdal_vector_set_field_type command line option, 447
gdal_vector_set_geom_type command line option, 452
gdal_vector_simplify command line option, 459
gdal_vector_simplify_coverage command line option, 463
gdal_vector_sql command line option, 467
gdal_vector_swap_xy command line option, 471
gdal_vector_write command line option, 476
gdal_vsi_copy command line option, 496
--skip-geolocation
gdal_raster_compare command line option, 129
--skip-geotransform
gdal_raster_compare command line option, 129
--skip-metadata
gdal_raster_compare command line option, 129
--skip-nodata
gdal_raster_as_features command line option, 93
--skip-on-type-mismatch
gdal_vector_explode_collections command line option, 365
--skip-overview
gdal_raster_compare command line option, 129
--skip-rpc
gdal_raster_compare command line option, 129
--skip-subdataset
gdal_raster_compare command line option, 130
--smoothing
gdal_vector_grid command line option, 369, 370
--smoothing-iterations
gdal_raster_fill_nodata command line option, 148
--snapping-distance
gdal_vector_clean_coverage command line option, 306
--source
gdal_dataset_copy command line option, 493
gdal_dataset_rename command line option, 494
--source-crs-field-name
gdal_raster_index command line option, 156
gdal_vector_index command line option, 377
--source-crs-format
gdal_raster_index command line option, 156
gdal_vector_index command line option, 377
--source-layer-field-content
gdal_vector_concat command line option, 321
--source-layer-field-name
gdal_vector_concat command line option, 321
--source-layer-index
gdal_vector_index command line option, 377
--source-layer-name
gdal_vector_index command line option, 377
--sozip-chunk-size
gdal_vsi_sozip command line option, 504, 505
sozip command line option, 691
--sozip-min-file-size
gdal_vsi_sozip command line option, 504, 505
sozip command line option, 691
--spatial-extent-adjustment
gdal_raster_pansharpen command line option, 179
--spectral
gdal_raster_pansharpen command line option, 180

--split-multipolygons
gdal_raster_footprint command line option, 145
--sql
gdal_vector_grid command line option, 368
gdal_vector_info command line option, 352
gdal_vector_rasterize command line option, 424
gdal_vector_sql command line option, 466
--srcnodata
gdal2tiles command line option, 524
--start-angle
gdal_raster_viewshed command line option, 275
--stat
gdal_raster_zonal_stats command line option, 282
--stats
gdal_mdim_info command line option, 479
gdal_raster_edit command line option, 140
gdal_raster_info command line option, 84
--strategy
gdal_raster_fill_nodata command line option, 148
gdal_raster_zonal_stats command line option, 282
gdal_vsi_sync command line option, 501
--strict
gdal_mdim_convert command line option, 484
--subdataset
gdal_raster_info command line option, 84
--subset
gdal_mdim_convert command line option, 484
--summary
gdal_mdim_info command line option, 479
gdal_vector_info command line option, 352
--tap
gdal_vector_rasterize command line option, 424
--target-aligned-pixels
gdal_raster_mosaic command line option, 162
gdal_raster_reproject command line option, 226
gdal_raster_stack command line option, 253
gdal_vector_rasterize command line option, 424
--target-height
gdal_raster_viewshed command line option, 275
--target-values
gdal_raster_proximity command line option, 216
--tight
gdal_vector_concave_hull command line option, 325
--tile-size
gdal_raster_tile command line option, 258
--tiledriver
gdal2tiles command line option, 525
--tilesize
gdal2tiles command line option, 525
--tiling-scheme
gdal_raster_tile command line option, 258
--title
gdal2tiles command line option, 526
gdal_raster_tile command line option, 259
--tmscompatible
gdal2tiles command line option, 524
--to
gdal_raster_reproject command line option, 226
gdal_raster_update command line option, 269
--tolerance
gdal_vector_simplify command line option, 457
gdal_vector_simplify_coverage command line option, 462
--transform-option
gdal_raster_reproject command line option, 226
gdal_raster_update command line option, 269
--transformer-option
gdal_vector_rasterize command line option, 425
--tree
gdal_vsi_list command line option, 499
--type
command line option, 531
--unit
gdal_raster_slope command line option, 246
--unset-color-table
gdal_raster_edit command line option, 140
--unset-fid
gdal_vector_edit command line option, 344
--unset-layer-metadata
gdal_vector_edit command line option, 344
--unset-metadata
gdal_raster_edit command line option, 140
gdal_vector_edit command line option, 344
--unset-metadata-domain
gdal_raster_edit command line option, 140

--update
command line option, 513
gdal_raster_as_features command line option, 94
gdal_raster_clean_collar command line option, 116
gdal_raster_index command line option, 157
gdal_raster_polygonize command line option, 214
gdal_raster_zonal_stats command line option, 283
gdal_vector_buffer command line option, 293
gdal_vector_check_coverage command line option, 298
gdal_vector_check_geometry command line option, 303
gdal_vector_clean_coverage command line option, 309
gdal_vector_clip command line option, 313
gdal_vector_combine command line option, 318
gdal_vector_concat command line option, 322
gdal_vector_concave_hull command line option, 326
gdal_vector_convert command line option, 330
gdal_vector_convex_hull command line option, 334
gdal_vector_dissolve command line option, 341
gdal_vector_edit command line option, 346
gdal_vector_explode_collections command line option, 366
gdal_vector_filter command line option, 349
gdal_vector_index command line option, 378
gdal_vector_layer_algebra command line option, 384
gdal_vector_make_point command line option, 388
gdal_vector_make_valid command line option, 391
gdal_vector_rasterize command line option, 425
gdal_vector_rename_layer command line option, 431
gdal_vector_reproject command line option, 435
gdal_vector_segmentize command line option, 444
gdal_vector_select command line option, 440
gdal_vector_set_field_type command line option, 448
gdal_vector_set_geom_type command line option, 453
gdal_vector_simplify command line option, 459
gdal_vector_simplify_coverage command line option, 463
gdal_vector_sql command line option, 467
gdal_vector_swap_xy command line option, 472
gdal_vector_write command line option, 476
--update-extent
gdal_vector_filter command line option, 348
--upsert
gdal_raster_zonal_stats command line option, 283
gdal_vector_buffer command line option, 293
gdal_vector_check_coverage command line option, 298
gdal_vector_check_geometry command line option, 303
gdal_vector_clean_coverage command line option, 309
gdal_vector_clip command line option, 313
gdal_vector_combine command line option, 318
gdal_vector_concat command line option, 322
gdal_vector_concave_hull command line option, 326
gdal_vector_convert command line option, 331
gdal_vector_convex_hull command line option, 334
gdal_vector_dissolve command line option, 341
gdal_vector_edit command line option, 346
gdal_vector_explode_collections command line option, 366
gdal_vector_filter command line option, 350
gdal_vector_make_point command line option, 388
gdal_vector_make_valid command line option, 392
gdal_vector_rename_layer command line option, 431
gdal_vector_reproject command line

option, 435
gdal_vector_segmentize command line option, 444
gdal_vector_select command line option, 440
gdal_vector_set_field_type command line option, 448
gdal_vector_set_geom_type command line option, 453
gdal_vector_simplify command line option, 459
gdal_vector_simplify_coverage command line option, 463
gdal_vector_sql command line option, 467
gdal_vector_swap_xy command line option, 472
gdal_vector_write command line option, 476
--url
gdal2tiles command line option, 526
gdal_raster_tile command line option, 259
--use-source-timestamp
gdal_raster_overview_refresh command line option, 177
--use-tempfile
gdal_vector_sort command line option, 455
--validate
sozip command line option, 690
--variant
gdal_raster_hillshade command line option, 151
--verbose
command line option, 510
gdal2tiles command line option, 524
gdal_vsi_sozip command line option, 507
sozip command line option, 690
--version
command line option, 519, 522
gdal2tiles command line option, 525
vector_common_options command line option, 655
--visible-value
gdal_raster_viewshed command line option, 275
--warp-option
gdal_raster_reproject command line option, 226
gdal_raster_update command line option, 270
--webp-lossless
gdal2tiles command line option, 527
--webp-quality
gdal2tiles command line option, 527
--webviewer
gdal2tiles command line option, 526
gdal_raster_tile command line option, 259
--weights
gdal_raster_pansharpen command line option, 180
gdal_raster_zonal_stats command line option, 282
--weights-band
gdal_raster_zonal_stats command line option, 282
--where
gdal_vector_filter command line option, 348
gdal_vector_info command line option, 352
gdal_vector_rasterize command line option, 425
--window
gdal_raster_clip command line option, 119
--wo
gdal_raster_reproject command line option, 226
gdal_raster_update command line option, 270
--x
gdal_vector_make_point command line option, 386
--xml-filename
command line option, 514
--xscale
gdal_raster_hillshade command line option, 151
gdal_raster_slope command line option, 246
--xyz
gdal2tiles command line option, 524
--y
gdal_vector_make_point command line option, 386
--yscale
gdal_raster_hillshade command line option, 152
gdal_raster_slope command line option, 246
--z
gdal_vector_make_point command line option, 386
--zero-for-flat
gdal_raster_aspect command line option, 96
--zfactor
gdal_raster_hillshade command line option, 152
--zfield
gdal_vector_grid command line option, 369

--zincrease
gdal_vector_grid command line option, 369
--zmultiply
gdal_vector_grid command line option, 369
--zones
gdal_raster_zonal_stats command line option, 282
--zones-band
gdal_raster_zonal_stats command line option, 282
--zones-layer
gdal_raster_zonal_stats command line option, 282
--zoom
gdal2tiles command line option, 524
-a
gdal2tiles command line option, 524
gdal_contour command line option, 534
gdal_grid command line option, 549
gdal_rasterize command line option, 564
gdal_vector_rasterize command line option, 423
-a_coord_epoch
command line option, 541
gdal_translate command line option, 574
ogr2ogr command line option, 669
-a_gt
gdal_translate command line option, 574
-a_nodata
command line option, 542
gdal_create command line option, 539
gdal_merge command line option, 556
gdal_rasterize command line option, 565
gdal_translate command line option, 574
gdal_viewshed command line option, 578
-a_offset
gdal_translate command line option, 574
-a_scale
gdal_translate command line option, 574
-a_srs
command line option, 521, 541
gdal_create command line option, 539
gdal_grid command line option, 548
gdal_rasterize command line option, 565
gdal_translate command line option, 573
gdalbuildvrt command line option, 591
ogr2ogr command line option, 669
ogr_layer_algebra command line option, 686
ogrmerge command line option, 683
-a_ullr
command line option, 541
gdal_create command line option, 539
gdal_translate command line option, 574
-a_ulurll
command line option, 541
-accept_different_schemas
ogrtindex command line option, 678
-add
gdal_rasterize command line option, 564
-addalpha
gdalbuildvrt command line option, 589
-addfields
ogr2ogr command line option, 674
-al
ogrinfo command line option, 657
-alg
command line option, 597–599, 601
nearblack command line option, 646
-allbands
gdal2xyz command line option, 528
-allow_projection_difference
gdalbuildvrt command line option, 590
-alo
gnmanalyse command line option, 689
-alpha
command line option, 600
-alt
command line option, 598
-amax
gdal_contour command line option, 534
-amin
gdal_contour command line option, 534
-append
ogr2ogr command line option, 667
ogrmerge command line option, 682
-approx_stats
command line option, 541
gdalinfo command line option, 605
-array
gdalmdiminfo command line option, 649
gdalmdimtranslate command line option, 653
-arrayoption
gdalmdiminfo command line option, 649
gdalmdimtranslate command line option, 653
-at
gdal_rasterize command line option, 564
-az
command line option, 598
-b
command line option, 544, 558, 596
gdal2tiles command line option, 526
gdal2xyz command line option, 528
gdal_contour command line option, 534
gdal_footprint command line option, 545
gdal_polygonize command line option, 560

gdal_raster_as_features command line option, 93
gdal_raster_aspect command line option, 96
gdal_raster_color_map command line option, 122
gdal_raster_contour command line option, 126
gdal_raster_edit command line option, 138
gdal_raster_fill_nodata command line option, 148
gdal_raster_footprint command line option, 144
gdal_raster_hillshade command line option, 151
gdal_raster_mosaic command line option, 161
gdal_raster_pixel_info command line option, 206
gdal_raster_polygonize command line option, 213
gdal_raster_proximity command line option, 216
gdal_raster_roughness command line option, 235
gdal_raster_scale command line option, 238
gdal_raster_sieve command line option, 249
gdal_raster_slope command line option, 246
gdal_raster_stack command line option, 252
gdal_raster_tpi command line option, 262
gdal_raster_tri command line option, 265
gdal_raster_viewshed command line option, 273
gdal_raster_zonal_stats command line option, 281
gdal_rasterize command line option, 563
gdal_translate command line option, 571
gdal_vector_rasterize command line option, 423
gdal_viewshed command line option, 578
gdaladdo command line option, 582
gdalbuildvrt command line option, 590
gdallocationinfo command line option, 614
gdalwarp command line option, 633
pct2rgb command line option, 647

-band
gdal2xyz command line option, 528

-bandcount
gdaltindex command line option, 627

-bands
gdal_create command line option, 539

-bitdepth
command line option, 558

-bl
gnmmanage command line option, 687

-burn
gdal_create command line option, 539
gdal_rasterize command line option, 564

-c
gdal2tiles command line option, 526
gdal_raster_polygonize command line option, 213
gdal_raster_sieve command line option, 249
gnmmanage command line option, 687

-cblend
gdalwarp command line option, 639

-cc
gdal_viewshed command line option, 578

-checksum
gdalinfo command line option, 606

-cl
gdalwarp command line option, 639

-clean
gdaladdo command line option, 582

-clipdst
ogr2ogr command line option, 672

-clipdstlayer
ogr2ogr command line option, 672

-clipdstsql
ogr2ogr command line option, 672

-clipdstwhere
ogr2ogr command line option, 672

-clipsrc
gdal_grid command line option, 549
ogr2ogr command line option, 672

-clipsrclayer
gdal_grid command line option, 549
ogr2ogr command line option, 672

-clipsrcsql
gdal_grid command line option, 549
ogr2ogr command line option, 672

-clipsrcwhere
gdal_grid command line option, 549
ogr2ogr command line option, 672

-co
command line option, 521, 558, 596
gdal_create command line option, 539
gdal_grid command line option, 549
gdal_merge command line option, 555
gdal_proximity command line option, 562
gdal_rasterize command line option, 565
gdal_retile command line option, 567
gdal_translate command line option, 575

gdal_viewshed command line option, 578
gdalbuildvrt command line option, 591
gdalenhance command line option, 602
gdalmdimtranslate command line option, 653
gdalwarp command line option, 638
nearblack command line option, 645
-color
nearblack command line option, 645
-colorinterp
gdal_translate command line option, 574
gdaltindex command line option, 627
-colorinterp_X
gdal_translate command line option, 574
-colorinterp_<X>
command line option, 542
-combine_bands
gdal_footprint command line option, 545
-combined
command line option, 598
-compute_edges
command line option, 596
-config
gdalenhance command line option, 603
-convex_hull
gdal_footprint command line option, 545
-create
command line option, 679
-createonly
gdal_merge command line option, 556
-crop_to_cutline
gdalwarp command line option, 639
-csql
gdalwarp command line option, 639
-csv
gdal2xyz command line option, 528
gdal_retile command line option, 568
-csvDelim
gdal_retile command line option, 568
-ct
gdaltransform command line option, 630
gdalwarp command line option, 635
ogr2ogr command line option, 671
-ct_opt
ogr2ogr command line option, 671
-cutline
gdalwarp command line option, 639
-cutline_srs
gdalwarp command line option, 639
-cvmd
gdalwarp command line option, 639
-cwhere
gdalwarp command line option, 639
-d
gdal2tiles command line option, 524
gdal_raster_reproject command line option, 225
gdal_vector_concat command line option, 320
gdal_vector_reproject command line option, 433
gnmanalyse command line option, 688
-dateTimeTo
ogr2ogr command line option, 673
-datelineoffset
ogr2ogr command line option, 672
-densify
gdal_footprint command line option, 546
-detailed
gdalmdiminfo command line option, 649
-dialect
gdal_rasterize command line option, 564
ogr2ogr command line option, 668
ogrinfo command line option, 658
-dim
ogr2ogr command line option, 669
-dir
gnmmanage command line option, 687
-distunits
gdal_proximity command line option, 562
-dmo
gdal_translate command line option, 575
-doo
gdalwarp command line option, 639
ogr2ogr command line option, 671
-ds_transaction
ogr2ogr command line option, 672
-dsco
command line option, 679
gdal_contour command line option, 535
gdal_footprint command line option, 546
gnmanalyse command line option, 689
gnmmanage command line option, 687
ogr2ogr command line option, 669
ogr_layer_algebra command line option, 685
ogrmerge command line option, 682
-dstalpha
gdalwarp command line option, 638
-dstband
gdal_proximity command line option, 562
gdalwarp command line option, 634
-dstnodata
gdal2xyz command line option, 529
gdalwarp command line option, 638
-dumpdiffs
gdalcompare command line option, 592
-e

gdal2tiles command line option, 524
gdal_contour command line option, 535
gdal_raster_contour command line option, 126
gdalsrsinfo command line option, 620
-eco
gdal_translate command line option, 573
-emptyStrAsNull
ogr2ogr command line option, 674
-epo
gdal_translate command line option, 573
-equalize
gdalenhance command line option, 603
-et
command line option, 619
gdalwarp command line option, 636
-exact_color_entry
command line option, 600
-expand
gdal_translate command line option, 571
-explodecollections
ogr2ogr command line option, 674
-exponent
gdal_translate command line option, 572
-extent3D
ogrinfo command line option, 659
-f
command line option, 512, 617, 679
gdal_contour command line option, 535
gdal_dataset_copy command line option, 493
gdal_dataset_delete command line option, 494
gdal_dataset_identify command line option, 490
gdal_dataset_rename command line option, 494
gdal_external command line option, 78
gdal_mdim_convert command line option, 483
gdal_mdim_mosaic command line option, 487
gdal_polygonize command line option, 560
gdal_raster_as_features command line option, 94
gdal_raster_aspect command line option, 97
gdal_raster_blend command line option, 99
gdal_raster_calc command line option, 113
gdal_raster_clean_collar command line option, 116
gdal_raster_clip command line option, 119
gdal_raster_color_map command line option, 123
gdal_raster_contour command line option, 127
gdal_raster_convert command line option, 132
gdal_raster_create command line option, 135
gdal_raster_fill_nodata command line option, 149
gdal_raster_footprint command line option, 146
gdal_raster_hillshade command line option, 153
gdal_raster_index command line option, 156
gdal_raster_info command line option, 83
gdal_raster_materialize command line option, 158
gdal_raster_mosaic command line option, 163
gdal_raster_neighbors command line option, 167
gdal_raster_nodata_to_alpha command line option, 169
gdal_raster_pansharpen command line option, 180
gdal_raster_pixel_info command line option, 206
gdal_raster_polygonize command line option, 213
gdal_raster_proximity command line option, 217
gdal_raster_reclassify command line option, 221
gdal_raster_reproject command line option, 227
gdal_raster_resize command line option, 231
gdal_raster_rgb_to_palette command line option, 234
gdal_raster_roughness command line option, 236
gdal_raster_scale command line option, 239
gdal_raster_select command line option, 242
gdal_raster_set_type command line option, 244
gdal_raster_sieve command line option, 250
gdal_raster_slope command line option, 247
gdal_raster_stack command line option, 253
gdal_raster_tile command line option, 260

gdal_raster_tpi command line option, 263
gdal_raster_tri command line option, 265
gdal_raster_unscale command line option, 268
gdal_raster_viewshed command line option, 275
gdal_raster_write command line option, 277
gdal_raster_zonal_stats command line option, 283
gdal_vector_buffer command line option, 293
gdal_vector_check_coverage command line option, 298
gdal_vector_check_geometry command line option, 303
gdal_vector_clean_coverage command line option, 308
gdal_vector_clip command line option, 313
gdal_vector_combine command line option, 317
gdal_vector_concat command line option, 322
gdal_vector_concave_hull command line option, 326
gdal_vector_convert command line option, 330
gdal_vector_convex_hull command line option, 334
gdal_vector_dissolve command line option, 341
gdal_vector_edit command line option, 345
gdal_vector_explode_collections command line option, 366
gdal_vector_filter command line option, 349
gdal_vector_grid command line option, 368
gdal_vector_index command line option, 378
gdal_vector_info command line option, 352
gdal_vector_layer_algebra command line option, 384
gdal_vector_make_point command line option, 387
gdal_vector_make_valid command line option, 391
gdal_vector_partition command line option, 397
gdal_vector_rasterize command line option, 425
gdal_vector_rename_layer command line option, 430
gdal_vector_reproject command line option, 434
gdal_vector_segmentize command line option, 443
gdal_vector_select command line option, 439
gdal_vector_set_field_type command line option, 447
gdal_vector_set_geom_type command line option, 452
gdal_vector_simplify command line option, 458
gdal_vector_simplify_coverage command line option, 463
gdal_vector_sql command line option, 467
gdal_vector_swap_xy command line option, 471
gdal_vector_write command line option, 475
gdal_vsi_list command line option, 499
gnmanalyse command line option, 689
gnmmanage command line option, 687
ogr2ogr command line option, 667
ogr_layer_algebra command line option, 685
ogrmerge command line option, 682
ogrtindex command line option, 677
-features
ogrinfo command line option, 657
-fetch_md
gdaltindex command line option, 628
-fid
ogr2ogr command line option, 671
ogrinfo command line option, 658
-fieldTypeToString
ogr2ogr command line option, 673
-field_sep
gdallocationinfo command line option, 615
gdaltransform command line option, 631
-field_strategy
ogrmerge command line option, 683
-fielddomain
ogrinfo command line option, 658
-fieldmap
ogr2ogr command line option, 674
-fields
ogrinfo command line option, 658
-filename_filter
gdaltindex command line option, 625
-fixed-buf-val
gdal_proximity command line option, 562
-fl
gdal_contour command line option, 535
-forceNullable
ogr2ogr command line option, 674
-fr

command line option, 617
-g
gdal2tiles command line option, 526
sozip command line option, 690
-gcp
command line option, 542
gdal_translate command line option, 575
gdaltransform command line option, 630
ogr2ogr command line option, 674
-geoloc
gdallocationinfo command line option, 615
gdaltransform command line option, 630
gdalwarp command line option, 636
-geom
ogrinfo command line option, 658
-geomfield
ogr2ogr command line option, 668
ogrinfo command line option, 658
-get_coord
command line option, 680
-get_pos
command line option, 680
-get_subline
command line option, 680
-group
gdalmdimtranslate command line option, 653
-gt
gdal_contour command line option, 535
ogr2ogr command line option, 672
-gti_filename
gdaltindex command line option, 627
-h
gdal2tiles command line option, 525
gdal2xyz command line option, 529
-hidenodata
gdalbuildvrt command line option, 589
-hist
gdalinfo command line option, 605
-i
gdal_contour command line option, 535
gdal_mdim_mosaic command line option, 487
gdal_raster_calc command line option, 112
gdal_raster_create command line option, 134
gdal_raster_pansharpen command line option, 179
gdal_raster_tile command line option, 257
gdal_rasterize command line option, 563
gdal_vector_create command line option, 337
gdal_vsi_sozip command line option, 503, 505–507
gdaltransform command line option, 630
-ic
gnmmanage command line option, 687
-if
gdal_create command line option, 539
gdal_translate command line option, 571
gdalinfo command line option, 606
gdalmdiminfo command line option, 649
gdalmdimtranslate command line option, 653
gdalwarp command line option, 638
ogr2ogr command line option, 667
ogrinfo command line option, 657
-ignore_extra_input
gdallocationinfo command line option, 615
gdaltransform command line option, 630
-ignore_srcmaskband
gdalbuildvrt command line option, 590
-igor
command line option, 598
-info
gnmmanage command line option, 687
-init
gdal_merge command line option, 556
gdal_rasterize command line option, 565
-inodata
gdal_contour command line option, 535
-input_ds
ogr_layer_algebra command line option, 685
-input_fields
ogr_layer_algebra command line option, 686
-input_file_list
gdalbuildvrt command line option, 591
-input_lyr
ogr_layer_algebra command line option, 685
-interp
command line option, 544
-iv
gdal_viewshed command line option, 579
-j
gdal_raster_pansharpen command line option, 179
gdal_raster_reproject command line option, 224
gdal_raster_tile command line option, 257
gdal_raster_viewshed command line option, 274
gdal_viewshed command line option, 580
gdal_vsi_sozip command line option, 503
gdal_vsi_sync command line option, 501
sozip command line option, 690
-json

gdalinfo command line option, 605
ogrinfo command line option, 657
-k
gdal2tiles command line option, 526
-l
command line option, 513, 679
gdal_grid command line option, 549
gdal_raster_pixel_info command line option, 206
gdal_rasterize command line option, 564
gdal_vector_export_schema command line option, 361
gdal_vector_grid command line option, 368
gdal_vector_info command line option, 352
gdal_vector_rasterize command line option, 423
gdal_vsi_list command line option, 499
gnmanalyse command line option, 689
gnmmanage command line option, 687
sozip command line option, 690
-l_srs
gdallocationinfo command line option, 615
-lco
command line option, 513, 679
gdal_contour command line option, 535
gdal_footprint command line option, 546
gdal_polygonize command line option, 560
gdaltindex command line option, 626
gnmanalyse command line option, 689
ogr2ogr command line option, 669
ogr_layer_algebra command line option, 685
ogrmerge command line option, 682
-levels
gdal_retile command line option, 568
-lf
command line option, 679
-lifonly
gdallocationinfo command line option, 614
-limit
gdalmdiminfo command line option, 649
ogr2ogr command line option, 671
ogrinfo command line option, 657
-listmdd
gdalinfo command line option, 606
ogrinfo command line option, 659
-ln
command line option, 679
-lname
ogrtindex command line option, 677
-lnum
ogrtindex command line option, 677
-location_field_name
gdal_footprint command line option, 546
-location-name
command line option, 513
-lyr_name
gdal_footprint command line option, 546
gdaltindex command line option, 626
-m
command line option, 680
gdal_raster_reclassify command line option, 221
-mRes
ogr2ogr command line option, 670
-makevalid
ogr2ogr command line option, 673
-mapFieldType
ogr2ogr command line option, 673
-mask
command line option, 544
gdal_polygonize command line option, 560
gdal_translate command line option, 571
gdaltindex command line option, 628
-max_pixel_size
gdaltindex command line option, 625
-max_points
gdal_footprint command line option, 546
-maxdist
gdal_proximity command line option, 562
-maxsubfields
ogr2ogr command line option, 674
-mb
command line option, 680
-md
command line option, 544
gdal_viewshed command line option, 578
-mdd
gdalinfo command line option, 606
ogrinfo command line option, 659
-me
command line option, 680
-method_ds
ogr_layer_algebra command line option, 685
-method_fields
ogr_layer_algebra command line option, 686
-method_lyr
ogr_layer_algebra command line option, 685
-min_pixel_size
gdaltindex command line option, 625
-min_ring_area
gdal_footprint command line option, 546
-minsize
gdaladdo command line option, 582
-mm

gdalinfo command line option, 605
-mo
command line option, 542
gdal_create command line option, 539
gdal_translate command line option, 574
gdaltindex command line option, 628
ogr2ogr command line option, 675
-multi
gdalwarp command line option, 638
-multidirectional
command line option, 598
-n
gdal2tiles command line option, 526
gdal_merge command line option, 556
rgb2pct command line option, 648
-nb
nearblack command line option, 645
-near
nearblack command line option, 645
-nearest_color_entry
command line option, 600
-nln
gdal_contour command line option, 535
ogr2ogr command line option, 669
ogrmerge command line option, 682
-nlt
ogr2ogr command line option, 669
ogr_layer_algebra command line option, 686
-noNativeData
ogr2ogr command line option, 675
-no_location
gdal_footprint command line option, 546
-nocount
ogrinfo command line option, 659
-noct
gdalinfo command line option, 605
-nodata
command line option, 558
gdal_proximity command line option, 562
gdaltindex command line option, 627
-nodata_max_mask_threshold
gdalbuildvrt command line option, 590
-noextent
ogrinfo command line option, 659
-nofl
gdalinfo command line option, 606
-nogcp
gdal_translate command line option, 575
gdalinfo command line option, 605
-nogeomtype
ogrinfo command line option, 659
-nomask
gdal_polygonize command line option, 560
gdalinfo command line option, 606
-nomd
gdalinfo command line option, 605
gdalwarp command line option, 639
ogr2ogr command line option, 675
ogrinfo command line option, 659
-non_strict
gdalbuildvrt command line option, 591
-nonodata
gdalinfo command line option, 605
-nopretty
gdalmdiminfo command line option, 649
-norat
gdal_translate command line option, 575
gdalinfo command line option, 605
-nosrcalpha
gdalwarp command line option, 638
-novshift
gdalwarp command line option, 635
-noxmp
gdal_translate command line option, 576
-o
command line option, 544, 680
gdal_dataset_identify command line option, 489
gdal_mdim_mosaic command line option, 487
gdal_merge command line option, 555
gdal_polygonize command line option, 560
gdal_raster_pixel_info command line option, 206
gdal_raster_tile command line option, 257
gdal_vector_export_schema command line option, 362
gdal_vsi_sozip command line option, 503, 505
gdalsrsinfo command line option, 620
nearblack command line option, 645
ogrmerge command line option, 682
-of
command line option, 520, 544, 558, 596, 680
gdal_create command line option, 539
gdal_footprint command line option, 546
gdal_grid command line option, 548
gdal_merge command line option, 555
gdal_proximity command line option, 562
gdal_rasterize command line option, 564
gdal_retile command line option, 567
gdal_translate command line option, 571
gdalattachpct command line option, 587
gdalenhance command line option, 602
gdalmdimtranslate command line option, 653
gdaltindex command line option, 625
gdalwarp command line option, 638

nearblack command line option, 645
ogr2ogr command line option, 667
pct2rgb command line option, 647
rgb2pct command line option, 648
-off
gdal_contour command line option, 535
-offset
command line option, 542
-om
gdal_viewshed command line option, 579
-on
command line option, 680
-oo
command line option, 542
gdal_footprint command line option, 546
gdal_grid command line option, 549
gdal_rasterize command line option, 565
gdal_translate command line option, 576
gdaladdo command line option, 582
gdalbuildvrt command line option, 591
gdalinfo command line option, 606
gdallocationinfo command line option, 615
gdalmdiminfo command line option, 649
gdalmdimtranslate command line option, 654
gdalwarp command line option, 639
ogr2ogr command line option, 671
ogrinfo command line option, 658
-opt
ogr_layer_algebra command line option, 685
-optim
gdal_rasterize command line option, 565
-order
gdaltransform command line option, 630
gdalwarp command line option, 635
ogr2ogr command line option, 674
-os
gdal_viewshed command line option, 579
-ot
gdal_create command line option, 538
gdal_grid command line option, 548
gdal_merge command line option, 555
gdal_proximity command line option, 562
gdal_rasterize command line option, 565
gdal_retile command line option, 568
gdal_translate command line option, 570
gdalenhance command line option, 603
gdaltindex command line option, 627
gdalwarp command line option, 637
-output_ds
ogr_layer_algebra command line option, 685
-output_lyr_name
ogr_layer_algebra command line option, 685
-output_xy
gdaltransform command line option, 630
-outsize
gdal_create command line option, 539
gdal_grid command line option, 548
gdal_translate command line option, 571
-ov
gdal_viewshed command line option, 579
-overlap
gdal_retile command line option, 568
-overview
gdallocationinfo command line option, 615
-overwrite
gdal_footprint command line option, 546
gdal_polygonize command line option, 561
gdalbuildvrt command line option, 591
gdaltindex command line option, 625
gdalwarp command line option, 639
ogr2ogr command line option, 667
ogr_layer_algebra command line option, 685
-overwrite_ds
ogrmerge command line option, 682
-overwrite_layer
ogrmerge command line option, 682
-ovr
gdal_footprint command line option, 545
gdal_translate command line option, 571
gdalwarp command line option, 636
-ox
gdal_viewshed command line option, 578
-oy
gdal_viewshed command line option, 578
-oz
gdal_viewshed command line option, 578
-p
command line option, 598, 679
gdal2tiles command line option, 524
gdal_contour command line option, 535
gdal_raster_contour command line option, 126
gdal_raster_pixel_info command line option, 207
gdal_raster_viewshed command line option, 274
-pct
gdal_merge command line option, 556
pct2rgb command line option, 647
rgb2pct command line option, 648
-pf
command line option, 680
-pixel-function

gdalbuildvrt command line option, 590
-pixel-function-arg
gdalbuildvrt command line option, 590
-pm
command line option, 680
-pn
command line option, 680
-preserve_fid
ogr2ogr command line option, 671
-progress
command line option, 679
ogr2ogr command line option, 668
ogrmerge command line option, 683
-proj4
gdalinfo command line option, 606
-projwin
gdal_translate command line option, 573
-projwin_srs
gdal_translate command line option, 573
-ps
gdal_merge command line option, 555
gdal_retile command line option, 568
-pyramidOnly
gdal_retile command line option, 568
-q
command line option, 508, 510, 543, 558, 596, 679
gdal2tiles command line option, 524
gdal_contour command line option, 534
gdal_create command line option, 538
gdal_footprint command line option, 545
gdal_grid command line option, 548
gdal_merge command line option, 556
gdal_polygonize command line option, 561
gdal_raster_convert command line option, 132
gdal_raster_reproject command line option, 227
gdal_rasterize command line option, 563
gdal_translate command line option, 570, 575
gdal_vector_convert command line option, 331
gdal_vector_reproject command line option, 435
gdal_vector_sql command line option, 468
gdal_viewshed command line option, 578
gdal_vsi_sozip command line option, 504, 505, 507
gdaladdo command line option, 582
gdalbuildvrt command line option, 589, 591
gdalmdimtranslate command line option, 653
gdaltindex command line option, 625
gdalwarp command line option, 633
nearblack command line option, 645
ogr2ogr command line option, 667
ogrinfo command line option, 657
-quiet
gdal_merge command line option, 556
-r
command line option, 558, 617, 680
gdal2tiles command line option, 524
gdal_dataset_identify command line option, 490
gdal_raster_overview_add command line option, 172
gdal_raster_overview_refresh command line option, 176
gdal_raster_pansharpen command line option, 179
gdal_raster_pixel_info command line option, 207
gdal_raster_reproject command line option, 225
gdal_raster_resize command line option, 230
gdal_raster_tile command line option, 258
gdal_raster_update command line option, 269
gdal_retile command line option, 568
gdal_translate command line option, 572
gdal_vsi_copy command line option, 496
gdal_vsi_delete command line option, 497
gdal_vsi_sozip command line option, 503
gdal_vsi_sync command line option, 501
gdaladdo command line option, 582
gdalbuildvrt command line option, 591
gdallocationinfo command line option, 614
gdalwarp command line option, 637
sozip command line option, 690
-recursive
gdaltindex command line option, 625
-refine_gcps
gdalwarp command line option, 636
-relaxedFieldNameMatch
ogr2ogr command line option, 674
-resolution
gdalbuildvrt command line option, 589
-resolveDomains
ogr2ogr command line option, 675
-resume
gdal_retile command line option, 568
-rgba
pct2rgb command line option, 647
-rl
ogrinfo command line option, 657
-rn

command line option, 680
-ro
command line option, 541
gdaladdo command line option, 582
ogrinfo command line option, 657
-rpc
gdaltransform command line option, 630
gdalwarp command line option, 636
-s
command line option, 597, 598, 680
gdal2tiles command line option, 524
gdal_raster_reproject command line option, 224
gdal_raster_sieve command line option, 249
gdal_vector_concat command line option, 320
gdal_vector_reproject command line option, 433
-s_coord_epoch
gdaltransform command line option, 630
gdalwarp command line option, 634
ogr2ogr command line option, 670
-s_srs
command line option, 521, 619
gdal_retile command line option, 568
gdaltransform command line option, 629
gdalwarp command line option, 634
ogr2ogr command line option, 670
ogrmerge command line option, 683
-scale
command line option, 542
gdal_translate command line option, 572
-scaleaxes
gdalmdimtranslate command line option, 654
-sd
gdalbuildvrt command line option, 590
gdalinfo command line option, 606
-sds
gdal_translate command line option, 575
gdalcompare command line option, 593
-segmentize
ogr2ogr command line option, 673
-select
ogr2ogr command line option, 668
-separate
gdal_merge command line option, 556
gdalbuildvrt command line option, 590
-setalpha
nearblack command line option, 645
-setci
gdalwarp command line option, 639
-setmask
nearblack command line option, 645
-setstats
command line option, 541
-si
command line option, 544
-simplify
gdal_footprint command line option, 546
ogr2ogr command line option, 672
-single
ogrmerge command line option, 682
-skip
gdal2xyz command line option, 528
-skip_binary
gdalcompare command line option, 592
-skip_different_projection
gdaltindex command line option, 626
ogrtindex command line option, 678
-skip_geolocation
gdalcompare command line option, 593
-skip_geotransform
gdalcompare command line option, 593
-skip_metadata
gdalcompare command line option, 593
-skip_overviews
gdalcompare command line option, 593
-skip_rpc
gdalcompare command line option, 593
-skip_srs
gdalcompare command line option, 593
-skipfailures
ogr2ogr command line option, 668
ogrmerge command line option, 683
-skipinvalid
ogr2ogr command line option, 673
-skipnodata
gdal2xyz command line option, 529
-snodata
gdal_contour command line option, 535
-so
ogrinfo command line option, 657
-spat
gdal_grid command line option, 549
ogr2ogr command line option, 668
ogrinfo command line option, 658
-spat_adjust
command line option, 558
-spat_srs
ogr2ogr command line option, 668
-split_polys
gdal_footprint command line option, 545
-splitlistfields
ogr2ogr command line option, 673
-sql
gdal_grid command line option, 549

gdal_rasterize command line option, 564
ogr2ogr command line option, 668
ogrinfo command line option, 658
-src_geom_type
ogrmerge command line option, 682
-src_layer_field_content
ogrmerge command line option, 683
-src_layer_field_name
ogrmerge command line option, 683
-src_srs_format
gdaltindex command line option, 626
ogrtindex command line option, 678
-src_srs_name
gdaltindex command line option, 626
ogrtindex command line option, 678
-srcalpha
gdalwarp command line option, 638
-srcband
gdal_proximity command line option, 562
gdalwarp command line option, 633
-srcnodata
gdal2xyz command line option, 529
gdal_footprint command line option, 545
gdalbuildvrt command line option, 589
gdalwarp command line option, 637
-srcwin
gdal2xyz command line option, 528
gdal_translate command line option, 573
-stats
command line option, 541
gdal_translate command line option, 575
gdalinfo command line option, 605
gdalmdiminfo command line option, 649
-strict
gdal_translate command line option, 571
gdalbuildvrt command line option, 591
gdalmdimtranslate command line option, 654
-subset
gdalmdimtranslate command line option, 654
-t
gdal2tiles command line option, 526
-t_coord_epoch
gdaltransform command line option, 630
gdalwarp command line option, 635
ogr2ogr command line option, 670
-t_cs
gdal_footprint command line option, 545
-t_srs
command line option, 521, 619
gdal_footprint command line option, 545
gdaltindex command line option, 626
gdaltransform command line option, 630
gdalwarp command line option, 634
gnmmanage command line option, 687
ogr2ogr command line option, 669
ogrmerge command line option, 683
ogrtindex command line option, 678
-tap
gdal_merge command line option, 556
gdal_rasterize command line option, 565
gdalbuildvrt command line option, 589
gdalwarp command line option, 636
-targetDir
gdal_retile command line option, 567
-te
gdal_rasterize command line option, 565
gdalbuildvrt command line option, 589
gdaltindex command line option, 627
gdalwarp command line option, 636
-te_srs
gdalwarp command line option, 636
-threads
command line option, 558
-tileIndex
gdal_retile command line option, 568
-tileIndexField
gdal_retile command line option, 568
-tileindex
gdalbuildvrt command line option, 589
gdaltindex command line option, 626
ogrtindex command line option, 678
-to
gdal_rasterize command line option, 565
gdaltransform command line option, 630
gdalwarp command line option, 635
-tps
gdaltransform command line option, 630
gdalwarp command line option, 635
ogr2ogr command line option, 674
-tr
command line option, 541
gdal_grid command line option, 548
gdal_rasterize command line option, 565
gdal_translate command line option, 571
gdalbuildvrt command line option, 589
gdaltindex command line option, 627
gdalwarp command line option, 636
-trigonometric
command line option, 599
-ts
gdal_rasterize command line option, 565
gdalwarp command line option, 636
-txe
gdal_grid command line option, 548
-tye
gdal_grid command line option, 548

-tz
    gdal_viewshed command line option, 578
-u
    command line option, 617
    gdal2tiles command line option, 526
-ul_lr
    gdal_merge command line option, 556
-unbl
    gnmmanage command line option, 688
-unblall
    gnmmanage command line option, 688
-units
    command line option, 542
-unscale
    gdal_translate command line option, 572
-unsetCoordPrecision
    ogr2ogr command line option, 670
-unsetDefault
    ogr2ogr command line option, 674
-unsetFid
    ogr2ogr command line option, 674
-unsetFieldWidth
    ogr2ogr command line option, 673
-unsetepoch
    command line option, 541
-unsetgt
    command line option, 541
-unsetmd
    command line option, 542
-unsetnodata
    command line option, 542
-unsetrpc
    command line option, 541
-unsetstats
    command line option, 541
-update
    ogr2ogr command line option, 668
    ogrmerge command line option, 682
-upsert
    ogr2ogr command line option, 667
-useDirForEachRow
    gdal_retile command line option, 568
-use_input_nodata
    gdal_proximity command line option, 562
-v
    command line option, 510
    gdal2tiles command line option, 524
    gdal_merge command line option, 556
    gdal_retile command line option, 568
    gdal_vsi_sozip command line option, 507
-valonly
    gdallocationinfo command line option, 614
-values
    gdal_proximity command line option, 562
-vrtnodata
    gdalbuildvrt command line option, 590
-vshift
    gdalwarp command line option, 635
-vv
    gdal_viewshed command line option, 579
-w
    command line option, 558
    gdal2tiles command line option, 526
-wgs84
    gdallocationinfo command line option, 615
-where
    gdal_grid command line option, 549
    gdal_rasterize command line option, 564
    ogr2ogr command line option, 668
    ogrinfo command line option, 658
-white
    nearblack command line option, 645
-wkt_format
    gdalinfo command line option, 606
    ogrinfo command line option, 659
-wm
    gdalwarp command line option, 638
-wo
    gdalwarp command line option, 637
-wrapdateline
    ogr2ogr command line option, 672
-write_absolute_path
    gdal_footprint command line option, 546
    gdalbuildvrt command line option, 591
    gdaltindex command line option, 626
    ogrtindex command line option, 678
-wt
    gdalwarp command line option, 637
-x
    command line option, 680
    gdal2tiles command line option, 524
-xml
    gdallocationinfo command line option, 614
-xscale
    command line option, 597, 598
-xyRes
    ogr2ogr command line option, 670
-y
    command line option, 680
-yscale
    command line option, 597, 599
-z
    command line option, 597
    gdal2tiles command line option, 524
    gdal_raster_hillshade command line option, 152
    gdal_raster_viewshed command line option, 273

-zRes
    ogr2ogr command line option, 670
-z_increase
    gdal_grid command line option, 548
-z_multiply
    gdal_grid command line option, 549
-zero_for_flat
    command line option, 599
-zfield
    gdal_grid command line option, 548
    ogr2ogr command line option, 674
<INPUT-CRS>
    gdal_vector_reproject command line option, 433
<OUTPUT-CRS>
    gdal_raster_reproject command line option, 225
<datasetname>
    command line option, 618
<datasource_name>
    ogrinfo command line option, 659
<dest_file>
    gdalattachpct command line option, 587
    pct2rgb command line option, 647
    rgb2pct command line option, 648
<dst_dataset_name>
    gdalwarp command line option, 640
    ogr2ogr command line option, 675
<dst_dataset>
    gdal2xyz command line option, 529
    gdal_translate command line option, 576
    gdalmdimtranslate command line option, 654
<dst_filename>
    gdal_footprint command line option, 546
    gdal_grid command line option, 549
    gdal_rasterize command line option, 566
<dstfile>
    command line option, 544
    gdal_proximity command line option, 562
    gdaltransform command line option, 631
<fieldname>
    gdal_polygonize command line option, 561
<file_or_dir>
    gdaltindex command line option, 626
<filename>
    gdaladdo command line option, 583
    sozip command line option, 691
<gnm_name>
    gnmanalyse command line option, 689
    gnmmanage command line option, 688
<golden_file>
    gdalcompare command line option, 593
<index_file>
    gdaltindex command line option, 626
<infile>
    nearblack command line option, 646
<input_dem>
    command line option, 596
<layer_name>
    ogr2ogr command line option, 675
<layer>
    gdal_polygonize command line option, 561
    gnmmanage command line option, 688
    ogrinfo command line option, 660
<levels>
    gdaladdo command line option, 583
<mode>
    command line option, 595, 617
    ogr_layer_algebra command line option, 684
<new_file>
    gdalcompare command line option, 593
<newdatasetname>
    command line option, 618
<out_dataset>
    command line option, 559
    gdal_create command line option, 539
<out_file>
    gdal_polygonize command line option, 561
<output_xxx_map>
    command line option, 596
<palette_file>
    gdalattachpct command line option, 587
<pan_dataset>
    command line option, 558
<raster_file>
    gdal_polygonize command line option, 560
<source_file>
    gdalattachpct command line option, 587
    pct2rgb command line option, 647
    rgb2pct command line option, 648
<spectral_dataset>[,band
    command line option, 559
<src_dataset_name>
    gdalwarp command line option, 639
    ogr2ogr command line option, 675
<src_dataset>
    gdal2xyz command line option, 529
    gdal_translate command line option, 576
    gdalmdimtranslate command line option, 654
<src_datasource>
    gdal_grid command line option, 549
    gdal_rasterize command line option, 566
<src_dsname>
    ogrmerge command line option, 682
<src_filename>

gdal_footprint command line option, 546
<srcfile>
command line option, 544
gdal_proximity command line option, 562
gdallocationinfo command line option, 615
gdaltransform command line option, 631
<srs_def>
gdalsrsinfo command line option, 621
<x>
gdallocationinfo command line option, 615
<y>
gdallocationinfo command line option, 615
<zip_filename>
sozip command line option, 691

A

A-Z]
command line option, 532
autoconnect
gnmmanage command line option, 687

B

built-in function
Run(), 31

C

change
gnmmanage command line option, 687
color_table
command line option, 532
color_text_file
command line option, 599
command line option
-A, 530
--A_band, 530
--NoDataValue, 530
--absolute-path, 513
--allBands, 532
--append, 513
--band-count, 514
--bbox, 514
--build, 520
--calc, 530
--cflags, 522
--classification, 1022
--co, 512, 532
--color-interpretation, 514
--color-table, 531
--config, 520
--contry-code, 1022
--creation-option, 512, 531
--datatype, 514
--debug, 520, 532
--extent, 531
--fetched-metadata, 514
--filename-filter, 513
--format, 512, 520, 531
--formats, 520, 522
--full-check, 508, 510
--help, 530, 541, 543, 558, 596, 619, 679
--help-general, 520, 530, 541, 543, 558, 596, 619, 679
--hideNoData, 530
--layer, 513
--libs, 522
--mask, 514
--max-pixel-size, 513
--metadata, 513
--min-pixel-size, 513
--nln, 513
--nodata, 514
--of, 512
--ogr-enabled, 522
--optfile, 520
--ot, 514
--outfile, 530
--output-crs, 513
--output-data-type, 514
--output-format, 512
--overwrite, 513, 532
--overwrite-layer, 513
--prefix, 522
--producer-id, 1022
--producer-name, 1022
--projectionCheck, 531
--projwin, 531
--quiet, 508, 510, 532, 596, 679
--recursive, 513
--resolution, 514
--scale, 1022
--type, 531
--update, 513
--verbose, 510
--version, 519, 522
--xml-filename, 514
-a_coord_epoch, 541
-a_nodata, 542
-a_srs, 521, 541
-a_ullr, 541
-a_ulurll, 541
-alg, 597–599, 601
-alpha, 600
-alt, 598
-approx_stats, 541
-az, 598
-b, 544, 558, 596
-bitdepth, 558
-co, 521, 558, 596

-colorinterp_<X>, 542
-combined, 598
-compute_edges, 596
-create, 679
-dsco, 679
-et, 619
-exact_color_entry, 600
-f, 512, 617, 679
-fr, 617
-gcp, 542
-get_coord, 680
-get_pos, 680
-get_subline, 680
-igor, 598
-interp, 544
-l, 513, 679
-lco, 513, 679
-lf, 679
-ln, 679
-location-name, 513
-m, 680
-mask, 544
-mb, 680
-md, 544
-me, 680
-mo, 542
-multidirectional, 598
-nearest_color_entry, 600
-nodata, 558
-o, 544, 680
-of, 520, 544, 558, 596, 680
-offset, 542
-on, 680
-oo, 542
-p, 598, 679
-pf, 680
-pm, 680
-pn, 680
-progress, 679
-q, 508, 510, 543, 558, 596, 679
-r, 558, 617, 680
-rn, 680
-ro, 541
-s, 597, 598, 680
-s_srs, 521, 619
-scale, 542
-setstats, 541
-si, 544
-spat_adjust, 558
-stats, 541
-t_srs, 521, 619
-threads, 558
-tr, 541
-trigonometric, 599
-u, 617
-units, 542
-unsetepoch, 541
-unsetgt, 541
-unsetmd, 542
-unsetnodata, 542
-unsetrpc, 541
-unsetstats, 541
-v, 510
-w, 558
-x, 680
-xscale, 597, 598
-y, 680
-yscale, 597, 599
-z, 597
-zero_for_flat, 599
<datasetname>, 618
<dstfile>, 544
<input_dem>, 596
<mode>, 595, 617
<newdatasetname>, 618
<out_dataset>, 559
<output_xxx_map>, 596
<pan_dataset>, 558
<spectral_dataset>[,band, 559
<srcfile>, 544
A-Z], 532
color_table, 532
color_text_file, 599
return_ds, 532
user_namespace, 532
connect
gnmmanage command line option, 687
create
gnmmanage command line option, 687

D

delete
gnmmanage command line option, 687
dijkstra
gnmanalyse command line option, 688
disconnect
gnmmanage command line option, 687

F

fields, 1731

G

gdal, 17
gdal convert, 64
gdal dataset, 488
gdal dataset check, 491
gdal dataset copy, 492
gdal dataset delete, 494

gdal dataset identify, 488
gdal dataset rename, 493
gdal driver cog validate, 507
gdal driver gpkg repack, 508
gdal driver gpkg validate, 509
gdal driver gti create, 510
gdal driver openfilegdb repack, 515
gdal driver parquet create-metadata-file, 516
gdal driver pdf list-layer, 517
gdal driver zarr add-georeferencing-convention, 518
gdal external, 77
gdal info, 36
gdal mdim, 477
gdal mdim convert, 482
gdal mdim info, 477
gdal mdim mosaic, 485
gdal pipeline, 64
gdal raster, 79
gdal raster as-features, 91
gdal raster aspect, 94
gdal raster blend, 97
gdal raster calc, 109
gdal raster clean-collar, 114
gdal raster clip, 116
gdal raster color-map, 121
gdal raster compare, 128
gdal raster contour, 124
gdal raster convert, 130
gdal raster create, 132
gdal raster edit, 136
gdal raster fill-nodata, 146
gdal raster footprint, 141
gdal raster hillshade, 149
gdal raster index, 153
gdal raster info, 81
gdal raster materialize, 157
gdal raster mosaic, 159
gdal raster neighbors, 163
gdal raster nodata-to-alpha, 168
gdal raster overview, 170
gdal raster overview add, 170
gdal raster overview delete, 174
gdal raster overview refresh, 175
gdal raster pansharpen, 178
gdal raster pipeline, 181
gdal raster pipeline read, 217
gdal raster pipeline write, 277
gdal raster pixel-info, 204
gdal raster polygonize, 211
gdal raster proximity, 214
gdal raster reclassify, 218
gdal raster reproject, 222
gdal raster resize, 229
gdal raster rgb-to-palette, 231
gdal raster roughness, 234
gdal raster scale, 236
gdal raster select, 240
gdal raster set_type, 242
gdal raster sieve, 248
gdal raster slope, 244
gdal raster stack, 250
gdal raster tile, 254
gdal raster tpi, 261
gdal raster tri, 263
gdal raster unscale, 266
gdal raster update, 268
gdal raster viewshed, 271
gdal raster zonal-stats, 278
gdal vector, 285
gdal vector buffer, 287
gdal vector check-coverage, 294
gdal vector check-geometry, 299
gdal vector clean-coverage, 304
gdal vector clip, 309
gdal vector combine, 314
gdal vector concat, 318
gdal vector concave-hull, 323
gdal vector convert, 328
gdal vector convex-hull, 331
gdal vector create, 335
gdal vector dissolve, 339
gdal vector edit, 342
gdal vector explode-collections, 363
gdal vector export-schema, 361
gdal vector filter, 346
gdal vector grid, 367
gdal vector index, 375
gdal vector info, 350
gdal vector layer-algebra, 379
gdal vector make-point, 385
gdal vector make-valid, 388
gdal vector materialize, 392
gdal vector partition, 394
gdal vector pipeline, 398
gdal vector pipeline read, 426
gdal vector pipeline write, 474
gdal vector rasterize, 421
gdal vector rename-layer, 427
gdal vector reproject, 432
gdal vector segmentize, 441
gdal vector select, 436
gdal vector set-field-type, 444
gdal vector set-geom-type, 448
gdal vector simplify, 455
gdal vector simplify-coverage, 460
gdal vector sort, 453
gdal vector sql, 464

gdal vector swap-xy, 469
gdal vector update, 472
gdal vsi, 495
gdal vsi copy, 496
gdal vsi delete, 497
gdal vsi list, 498
gdal vsi move, 499
gdal vsi sozip, 502
gdal vsi sync, 500
gdal2tiles, 522
gdal2tiles command line option
    ---jpeg-quality, 527
    --bingkey, 526
    --copyright, 526
    --exclude, 524
    --excluded-values, 525
    --excluded-values-pct-threshold, 525
    --force-kml, 526
    --googlekey, 526
    --help, 523, 525
    --help-general, 524
    --legacy, 524
    --mapml-template, 526
    --mpi, 524
    --no-kml, 526
    --nodata-values-pct-threshold, 525
    --processes, 524
    --profile, 524
    --quiet, 524
    --resampling, 524
    --resume, 524
    --s_srs, 524
    --srcnodata, 524
    --tiledriver, 525
    --tilesize, 525
    --title, 526
    --tmscompatible, 524
    --url, 526
    --verbose, 524
    --version, 525
    --webp-lossless, 527
    --webp-quality, 527
    --webviewer, 526
    --xyz, 524
    --zoom, 524
    -a, 524
    -b, 526
    -c, 526
    -d, 524
    -e, 524
    -g, 526
    -h, 525
    -k, 526
    -n, 526
    -p, 524
    -q, 524
    -r, 524
    -s, 524
    -t, 526
    -u, 526
    -v, 524
    -w, 526
    -x, 524
    -z, 524
gdal2xyz, 528
gdal2xyz command line option
    --help, 528, 529
    --help-general, 528
    -allbands, 528
    -b, 528
    -band, 528
    -csv, 528
    -dstnodata, 529
    -h, 529
    -skip, 528
    -skipnodata, 529
    -srcnodata, 529
    -srcwin, 528
    <dst_dataset>, 529
    <src_dataset>, 529
gdal_calc, 529
gdal_contour, 534
gdal_contour command line option
    -3d, 535
    --help, 534
    --help-general, 534
    --quiet, 534
    -a, 534
    -amax, 534
    -amin, 534
    -b, 534
    -dsco, 535
    -e, 535
    -f, 535
    -fl, 535
    -gt, 535
    -i, 535
    -inodata, 535
    -lco, 535
    -nln, 535
    -off, 535
    -p, 535
    -q, 534
    -snodata, 535
gdal_create, 538
gdal_create command line option
    --help, 538
    --help-general, 538

--quiet, 538
-a_nodata, 539
-a_srs, 539
-a_ullr, 539
-bands, 539
-burn, 539
-co, 539
-if, 539
-mo, 539
-of, 539
-ot, 538
-outsize, 539
-q, 538
<out_dataset>, 539
gdal_dataset_check command line option
--input, 492
gdal_dataset_copy command line option
---format, 493
--destination, 493
--overwrite, 493
--source, 493
-f, 493
gdal_dataset_delete command line option
---format, 494
--filename, 494
-f, 494
gdal_dataset_identify command line option
--detailed, 490
--filename, 489
--force-recursive, 490
--format, 490
--of, 490
--output, 489
--output-format, 490
--recursive, 490
--report-failures, 490
-f, 490
-o, 489
-r, 490
gdal_dataset_rename command line option
---format, 494
--destination, 494
--overwrite, 494
--source, 494
-f, 494
gdal_driver_zarr_add_georeferencing_convention command line option
--convention, 518
gdal_edit, 540
gdal_external command line option
--command, 78
--format, 78
--if, 78
--input-format, 78
--of, 78
--output-format, 78
-f, 78
gdal_fillnodata, 543
gdal_footprint, 544
gdal_footprint command line option
--help, 545
--help-general, 545
--quiet, 545
-b, 545
-combine_bands, 545
-convex_hull, 545
-densify, 546
-dsco, 546
-lco, 546
-location_field_name, 546
-lyr_name, 546
-max_points, 546
-min_ring_area, 546
-no_location, 546
-of, 546
-oo, 546
-overwrite, 546
-ovr, 545
-q, 545
-simplify, 546
-split_polys, 545
-srcnodata, 545
-t_cs, 545
-t_srs, 545
-write_absolute_path, 546
<dst_filename>, 546
<src_filename>, 546
gdal_grid, 547
gdal_grid command line option
--help, 548
--help-general, 548
--quiet, 548
-a, 549
-a_srs, 548
-clipsrc, 549
-clipsrclayer, 549
-clipsrcsql, 549
-clipsrcwhere, 549
-co, 549
-l, 549
-of, 548
-oo, 549
-ot, 548
-outsize, 548
-q, 548
-spat, 549
-sql, 549
-tr, 548

-txe, 548
-tye, 548
-where, 549
-z_increase, 548
-z_multiply, 549
-zfield, 548
<dst_filename>, 549
<src_datasource>, 549
gdal_mdim_convert command line option
--array, 483
--array-option, 483
--co, 483
--creation-option, 483
--format, 483
--group, 483
--if, 484
--input-format, 484
--of, 483
--oo, 484
--open-option, 484
--output-format, 483
--overwrite, 483
--scale-axes, 484
--strict, 484
--subset, 484
-f, 483
gdal_mdim_info command line option
--array, 479
--array-option, 479
--detailed, 479
--if, 479
--input-format, 479
--limit, 479
--oo, 479
--open-option, 479
--stats, 479
--summary, 479
gdal_mdim_mosaic command line option
--array, 487
--co, 487
--creation-option, 487
--format, 487
--if, 487
--input, 487
--input-format, 487
--of, 487
--oo, 487
--open-option, 487
--output, 487
--output-format, 487
--overwrite, 487
-f, 487
-i, 487
-o, 487
gdal_merge, 554
gdal_merge command line option
--help, 555
--help-general, 555
-a_nodata, 556
-co, 555
-createonly, 556
-init, 556
-n, 556
-o, 555
-of, 555
-ot, 555
-pct, 556
-ps, 555
-q, 556
-quiet, 556
-separate, 556
-tap, 556
-ul_lr, 556
-v, 556
gdal_pansharpen, 557
gdal_polygonize, 559
gdal_polygonize command line option
-8, 560
--help, 560
--help-general, 560
-b, 560
-f, 560
-lco, 560
-mask, 560
-nomask, 560
-o, 560
-overwrite, 561
-q, 561
<fieldname>, 561
<layer>, 561
<out_file>, 561
<raster_file>, 560
gdal_proximity, 561
gdal_proximity command line option
--help, 561
--help-general, 561
-co, 562
-distunits, 562
-dstband, 562
-fixed-buf-val, 562
-maxdist, 562
-nodata, 562
-of, 562
-ot, 562
-srcband, 562
-use_input_nodata, 562
-values, 562
<dstfile>, 562

<srcfile>, 562
gdal_raster_as_features command line option
--append, 93
--band, 93
--co, 93
--creation-option, 93
--format, 94
--geometry-type, 93
--if, 93
--include-row-col, 93
--include-xy, 93
--input-format, 93
--layer-creation-option, 93
--lco, 93
--of, 94
--oo, 94
--open-option, 94
--output-format, 94
--output-layer, 93
--output-oo, 94
--output-open-option, 94
--overwrite, 94
--overwrite-layer, 94
--skip-nodata, 93
--update, 94
-b, 93
-f, 94
gdal_raster_aspect command line option
--append, 96
--band, 96
--co, 96
--convention, 96
--creation-option, 96
--format, 97
--gradient-alg, 96
--if, 97
--input-format, 97
--no-edges, 96
--of, 97
--oo, 97
--open-option, 97
--output-format, 97
--overwrite, 97
--zero-for-flat, 96
-b, 96
-f, 97
gdal_raster_blend command line option
--append, 99
--co, 99
--creation-option, 99
--format, 99
--if, 99
--input, 99
--input-format, 99
--of, 99
--oo, 99
--opacity, 99
--open-option, 99
--operator, 99
--output-format, 99
--overlay, 99
--overwrite, 100
-f, 99
gdal_raster_calc command line option
--append, 112
--calc, 111
--co, 112
--creation-option, 112
--datatype, 113
--dialect, 111
--flatten, 111
--format, 113
--if, 112
--input-format, 112
--no-check-crs, 112
--no-check-extent, 112
--nodata, 112
--of, 113
--oo, 112
--open-option, 112
--ot, 113
--output-data-type, 113
--output-format, 113
--overwrite, 113
--propagate-nodata, 112
-f, 113
-i, 112
gdal_raster_clean_collar command line option
--add-alpha, 115
--add-mask, 115
--algorithm, 115
--co, 116
--color, 115
--color-threshold, 115
--creation-option, 116
--format, 116
--of, 116
--output-format, 116
--overwrite, 116
--pixel-distance, 115
--update, 116
-f, 116
gdal_raster_clip command line option
--add-alpha, 118
--allow-bbox-outside-source, 118
--bbox, 118
--bbox-crs, 118

--co, 119
--creation-option, 119
--format, 119
--geometry, 118
--geometry-crs, 119
--if, 119
--input-format, 119
--like, 119
--like-layer, 119
--like-sql, 119
--like-where, 119
--of, 119
--only-bbox, 119
--oo, 120
--open-option, 120
--output-format, 119
--overwrite, 120
--window, 119
-f, 119
gdal_raster_color_map command line option
--add-alpha, 122
--band, 122
--co, 123
--color-map, 122
--color-selection, 123
--creation-option, 123
--format, 123
--of, 123
--output-format, 123
--overwrite, 123
-b, 122
-f, 123
gdal_raster_compare command line option
--if, 130
--input, 129
--input-format, 130
--oo, 130
--open-option, 130
--reference, 129
--skip-all-optional, 129
--skip-binary, 129
--skip-crs, 129
--skip-geolocation, 129
--skip-geotransform, 129
--skip-metadata, 129
--skip-overview, 129
--skip-rpc, 129
--skip-subdataset, 130
gdal_raster_contour command line option
--3d, 126
--band, 126
--co, 126
--creation-option, 126
--elevation-name, 126
--exp-base, 126
--format, 127
--group-transactions, 126
--if, 127
--input-format, 127
--input-nodata, 126
--interval, 126
--layer-creation-option, 127
--lco, 127
--levels, 126
--max-name, 126
--min-name, 126
--nln, 126
--of, 127
--off, 126
--offset, 126
--oo, 127
--open-option, 127
--output-format, 127
--output-layer, 126
--output-oo, 127
--output-open-option, 127
--overwrite, 127
--polygonize, 126
-b, 126
-e, 126
-f, 127
-p, 126
gdal_raster_convert command line option
--append, 131
--co, 131
--creation-option, 131
--format, 132
--if, 132
--input-format, 132
--of, 132
--oo, 132
--open-option, 132
--output-format, 132
--overwrite, 132
--quiet, 132
-f, 132
-q, 132
gdal_raster_create command line option
--append, 135
--band-count, 134
--bbox, 134
--burn, 134
--co, 135
--copy-metadata, 134
--copy-overviews, 134
--creation-option, 135
--crs, 134
--datatype, 135

--format, 135
--if, 135
--input, 134
--input-format, 135
--like, 134
--metadata, 134
--nodata, 134
--of, 135
--oo, 135
--open-option, 135
--ot, 135
--output-data-type, 135
--output-format, 135
--overwrite, 135
--size, 135
-f, 135
-i, 134
gdal_raster_edit command line option
--approx-stats, 137
--auxiliary, 138
--band, 138
--bbox, 138
--color-interpretation, 139
--color-map, 138
--crs, 138
--dataset, 139
--gcp, 139
--hist, 139
--metadata, 140
--nodata, 140
--offset, 139
--oo, 140
--open-option, 140
--scale, 139
--stats, 140
--unset-color-table, 140
--unset-metadata, 140
--unset-metadata-domain, 140
-b, 138
gdal_raster_fill_nodata command line option
--append, 148
--band, 148
--co, 148
--creation-option, 148
--format, 149
--if, 148
--input-format, 148
--mask, 148
--max-distance, 148
--of, 149
--oo, 148
--open-option, 148
--output-format, 149
--overwrite, 149
--smoothing-iterations, 148
--strategy, 148
-b, 148
-f, 149
gdal_raster_footprint command line option
--absolute-path, 144
--append, 145
--band, 144
--co, 145
--combine-bands, 144
--convex-hull, 144
--coordinate-system, 144
--creation-option, 145
--densify-distance, 144
--format, 146
--if, 145
--input-format, 145
--input-nodata, 144
--layer-creation-option, 145
--lco, 145
--location-field, 144
--max-points, 144
--min-ring-area, 144
--no-location-field, 145
--of, 146
--oo, 146
--open-option, 146
--output-crs, 145
--output-format, 146
--output-layer, 145
--overview, 145
--overwrite, 146
--simplify-tolerance, 145
--split-multipolygons, 145
-b, 144
-f, 146
gdal_raster_hillshade command line option
--altitude, 151
--append, 152
--azimuth, 151
--band, 151
--co, 152
--creation-option, 152
--format, 153
--gradient-alg, 151
--if, 152
--input-format, 152
--no-edges, 151
--of, 153
--oo, 152
--open-option, 152
--output-format, 153
--overwrite, 153
--variant, 151

--xscale, 151
--yscale, 152
--zfactor, 152
-b, 151
-f, 153
-z, 152
gdal_raster_index command line option
--absolute-path, 155
--append, 156
--base-url, 155
--co, 156
--creation-option, 156
--filename-filter, 155
--format, 156
--id-metadata-item, 155
--id-method, 155
--layer-creation-option, 156
--lco, 156
--location-name, 155
--max-pixel-size, 155
--metadata, 155
--min-pixel-size, 155
--nln, 156
--of, 156
--output-crs, 155
--output-format, 156
--output-layer, 156
--overwrite, 157
--overwrite-layer, 157
--profile, 155
--recursive, 156
--skip-errors, 156
--source-crs-field-name, 156
--source-crs-format, 156
--update, 157
-f, 156
gdal_raster_info command line option
--approx-stats, 83
--checksum, 83
--crs-format, 83
--format, 83
--hist, 83
--if, 84
--input-format, 84
--list-mdd, 83
--mdd, 83
--metadata-domain, 83
--min-max, 84
--mm, 84
--no-ct, 84
--no-fl, 84
--no-gcp, 84
--no-mask, 84
--no-md, 84
--no-nodata, 84
--of, 83
--oo, 84
--open-option, 84
--output-format, 83
--stats, 84
--subdataset, 84
-f, 83
gdal_raster_materialize command line option
--co, 158
--creation-option, 158
--format, 158
--of, 158
--output, 158
--output-format, 158
--overwrite, 158
-f, 158
gdal_raster_mosaic command line option
--absolute-path, 161
--add-alpha, 161
--append, 162
--band, 161
--bbox, 161
--co, 162
--creation-option, 162
--format, 163
--hide-nodata, 161
--if, 162
--input-format, 162
--input-nodata, 161
--of, 163
--oo, 163
--open-option, 163
--output-format, 163
--output-nodata, 161
--overwrite, 163
--pixel-function, 162
--pixel-function-arg, 162
--resolution, 162
--target-aligned-pixels, 162
-b, 161
-f, 163
gdal_raster_neighbors command line option
--append, 167
--band, 165
--co, 167
--creation-option, 167
--datatype, 167
--format, 167
--if, 167
--input-format, 167
--kernel, 165
--method, 166
--nodata, 166

--of, 167
--oo, 167
--open-option, 167
--ot, 167
--output-data-type, 167
--output-format, 167
--overwrite, 167
--size, 166
-f, 167
gdal_raster_nodata_to_alpha command line option
--co, 169
--creation-option, 169
--datatype, 169
--format, 169
--nodata, 169
--of, 169
--ot, 169
--output-data-type, 169
--output-format, 169
-f, 169
gdal_raster_overview_add command line option
--co, 171, 173
--creation-option, 173
--dataset, 171
--external, 172
--levels, 172
--min-size, 172
--oo, 173
--open-option, 173
--overview-src, 172
--resampling, 172
-r, 172
gdal_raster_overview_delete command line option
--dataset, 175
--external, 175
--oo, 175
--open-option, 175
gdal_raster_overview_refresh command line option
--bbox, 176
--dataset, 176
--external, 176
--levels, 176
--like, 176
--oo, 177
--open-option, 177
--resampling, 176
--use-source-timestamp, 177
-r, 176
gdal_raster_pansharpen command line option
--append, 180
--bit-depth, 179
--co, 180
--creation-option, 180
--format, 180
--if, 180
--input, 179
--input-format, 180
--nodata, 179
--num-threads, 179
--of, 180
--oo, 180
--open-option, 180
--output-format, 180
--overwrite, 180
--panchromatic, 179
--resampling, 179
--spatial-extent-adjustment, 179
--spectral, 180
--weights, 180
-f, 180
-i, 179
-j, 179
-r, 179
gdal_raster_pixel_info command line option
--co, 208
--creation-option, 208
--format, 206
--if, 207
--include-field, 206
--input-format, 207
--input-layer, 206
--layer, 206
--layer-creation-option, 208
--lco, 208
--of, 206
--oo, 207
--open-option, 207
--output, 206
--output-format, 206
--overview, 207
--overwrite, 208
--ovr, 207
--pos, 207
--position, 207
--position-crs, 207
--position-dataset, 206
--promote-pixel-value-to-z, 207
--resampling, 207
-b, 206
-f, 206
-l, 206
-o, 206
-p, 207
-r, 207

gdal_raster_polygonize command line option
--append, 213
--attribute-name, 213
--band, 213
--co, 213
--connect-diagonal-pixels, 213
--creation-option, 213
--format, 213
--if, 213
--input-format, 213
--layer-creation-option, 213
--lco, 213
--nln, 213
--of, 213
--oo, 214
--open-option, 214
--output-format, 213
--output-layer, 213
--output-oo, 214
--output-open-option, 214
--overwrite, 214
--overwrite-layer, 214
--update, 214
-b, 213
-c, 213
-f, 213
gdal_raster_proximity command line option
--append, 216
--band, 216
--co, 216
--creation-option, 216
--datatype, 217
--distance-units, 216
--fixed-value, 216
--format, 217
--if, 216
--input-format, 216
--max-distance, 216
--nodata, 216
--of, 217
--oo, 217
--open-option, 217
--ot, 217
--output-data-type, 217
--output-format, 217
--overwrite, 217
--target-values, 216
-b, 216
-f, 217
gdal_raster_read command line option
--if, 218
--input-format, 218
--oo, 218
--open-option, 218
gdal_raster_reclassify command line option
--append, 221
--co, 221
--creation-option, 221
--datatype, 221
--format, 221
--if, 221
--input-format, 221
--mapping, 221
--of, 221
--oo, 221
--open-option, 221
--ot, 221
--output-data-type, 221
--output-format, 221
--overwrite, 221
-f, 221
-m, 221
gdal_raster_reproject command line option
--add-alpha, 224
--append, 226
--bbox, 224
--bbox-crs, 224
--co, 226
--creation-option, 226
--error-threshold, 224
--et, 224
--format, 227
--if, 227
--input-crs, 224
--input-format, 227
--input-nodata, 224
--like, 224
--num-threads, 224
--of, 227
--oo, 227
--open-option, 227
--output-crs, 225
--output-format, 227
--output-nodata, 225
--overwrite, 227
--quiet, 227
--resampling, 225
--resolution, 226
--size, 226
--target-aligned-pixels, 226
--to, 226
--transform-option, 226
--warp-option, 226
--wo, 226
-d, 225
-f, 227
-j, 224
-q, 227

-r, 225
-s, 224
<OUTPUT-CRS>, 225
gdal_raster_resize command line option
--append, 230
--co, 231
--creation-option, 231
--format, 231
--if, 231
--input-format, 231
--of, 231
--oo, 231
--open-option, 231
--output-format, 231
--overwrite, 231
--resampling, 230
--resolution, 230
--size, 230
-f, 231
-r, 230
gdal_raster_rgb_to_palette command line option
--bit-depth, 233
--co, 233
--color-count, 233
--color-map, 233
--creation-option, 233
--format, 234
--no-dither, 233
--of, 234
--output-format, 234
--output-nodata, 233
--overwrite, 234
-f, 234
gdal_raster_roughness command line option
--append, 235
--band, 235
--co, 235
--creation-option, 235
--format, 236
--if, 236
--input-format, 236
--no-edges, 235
--of, 236
--oo, 236
--open-option, 236
--output-format, 236
--overwrite, 236
-b, 235
-f, 236
gdal_raster_scale command line option
--append, 239
--band, 238
--co, 239
--creation-option, 239
--datatype, 239
--exponent, 238
--format, 239
--if, 239
--input-format, 239
--input-max, 238
--input-min, 238
--no-clip, 239
--of, 239
--oo, 239
--open-option, 239
--ot, 239
--output-data-type, 239
--output-format, 239
--output-max, 238
--output-min, 239
--overwrite, 239
-b, 238
-f, 239
gdal_raster_select command line option
--append, 241
--band, 241
--co, 241
--creation-option, 241
--exclude, 241
--format, 242
--if, 242
--input-format, 242
--mask, 241
--of, 242
--oo, 242
--open-option, 242
--output-format, 242
--overwrite, 242
-f, 242
gdal_raster_set_type command line option
--co, 244
--creation-option, 244
--datatype, 244
--format, 244
--of, 244
--ot, 244
--output-data-type, 244
--output-format, 244
--overwrite, 244
-f, 244
gdal_raster_sieve command line option
--append, 249
--band, 249
--co, 249
--connect-diagonal-pixels, 249
--creation-option, 249
--format, 250

--if, 250
--input-format, 250
--mask, 249
--of, 250
--oo, 250
--open-option, 250
--output-format, 250
--overwrite, 250
--size-threshold, 249
-b, 249
-c, 249
-f, 250
-s, 249
gdal_raster_slope command line option
--append, 247
--band, 246
--co, 247
--creation-option, 247
--format, 247
--gradient-alg, 246
--if, 247
--input-format, 247
--no-edges, 246
--of, 247
--oo, 247
--open-option, 247
--output-format, 247
--overwrite, 247
--unit, 246
--xscale, 246
--yscale, 246
-b, 246
-f, 247
gdal_raster_stack command line option
--absolute-path, 252
--append, 253
--band, 252
--bbox, 252
--co, 253
--creation-option, 253
--format, 253
--hide-nodata, 252
--if, 253
--input-format, 253
--input-nodata, 252
--of, 253
--oo, 253
--open-option, 253
--output-format, 253
--output-nodata, 252
--overwrite, 253
--resolution, 252
--target-aligned-pixels, 253
-b, 252
-f, 253
gdal_raster_tile command line option
--add-alpha, 256
--aux-xml, 256
--co, 260
--convention, 256
--copy-src-metadata, 256
--copyright, 259
--creation-option, 260
--excluded-values, 259
--excluded-values-pct-threshold, 259
--format, 260
--if, 260
--input, 257
--input-format, 260
--kml, 257
--mapml-template, 260
--max-x, 257
--max-y, 257
--max-zoom, 257
--metadata, 257
--min-x, 257
--min-y, 257
--min-zoom, 257
--no-alpha, 257
--no-intersection-ok, 257
--nodata-values-pct-threshold, 259
--num-threads, 257
--of, 260
--oo, 260
--open-option, 260
--output, 257
--output-format, 260
--output-nodata, 257
--overview-resampling, 257
--parallel-method, 257
--resampling, 258
--resume, 258
--skip-blank, 258
--tile-size, 258
--tiling-scheme, 258
--title, 259
--url, 259
--webviewer, 259
-f, 260
-i, 257
-j, 257
-o, 257
-r, 258
gdal_raster_tpi command line option
--append, 262
--band, 262
--co, 262
--creation-option, 262

--format, 263
--if, 263
--input-format, 263
--no-edges, 262
--of, 263
--oo, 263
--open-option, 263
--output-format, 263
--overwrite, 263
-b, 262
-f, 263
gdal_raster_tri command line option
--algorithm, 265
--append, 265
--band, 265
--co, 265
--creation-option, 265
--format, 265
--if, 265
--input-format, 265
--no-edges, 265
--of, 265
--oo, 265
--open-option, 265
--output-format, 265
--overwrite, 265
-b, 265
-f, 265
gdal_raster_unscale command line option
--append, 267
--co, 267
--creation-option, 267
--datatype, 268
--format, 268
--if, 267
--input-format, 267
--of, 268
--oo, 267
--open-option, 267
--ot, 268
--output-data-type, 268
--output-format, 268
--overwrite, 268
-f, 268
gdal_raster_update command line option
--error-threshold, 269
--et, 269
--geometry, 269
--geometry-crs, 269
--if, 270
--input-format, 270
--no-update-overviews, 269
--oo, 270
--open-option, 270
--resampling, 269
--to, 269
--transform-option, 269
--warp-option, 270
--wo, 270
-r, 269
gdal_raster_viewshed command line option
--append, 275
--band, 273
--co, 275
--creation-option, 275
--curvature-coefficient, 273
--end-angle, 273
--format, 275
--height, 273
--high-pitch, 273
--if, 275
--input-format, 275
--invisible-value, 274
--low-pitch, 274
--max-distance, 274
--maybe-visible-value, 275
--min-distance, 274
--mode, 274
--num-threads, 274
--observer-spacing, 274
--of, 275
--oo, 275
--open-option, 275
--out-of-range-value, 274
--output-format, 275
--output-nodata, 274
--overwrite, 276
--pos, 274
--position, 274
--sd-filename, 275
--start-angle, 275
--target-height, 275
--visible-value, 275
-b, 273
-f, 275
-j, 274
-p, 274
-z, 273
gdal_raster_write command line option
--append, 278
--co, 277
--creation-option, 277
--format, 277
--of, 277
--output-format, 277
--overwrite, 277
-f, 277
gdal_raster_zonal_stats command line option

--append, 282
--band, 281
--chunk-size, 281
--co, 282
--creation-option, 282
--format, 283
--if, 283
--include-field, 282
--include-geom, 282
--input-format, 283
--layer-creation-option, 283
--lco, 283
--of, 283
--oo, 283
--open-option, 283
--output-format, 283
--output-layer, 283
--output-oo, 283
--output-open-option, 283
--overwrite, 283
--overwrite-layer, 283
--pixels, 282
--skip-errors, 283
--stat, 282
--strategy, 282
--update, 283
--upsert, 283
--weights, 282
--weights-band, 282
--zones, 282
--zones-band, 282
--zones-layer, 282
-b, 281
-f, 283
gdal_rasterize, 563
gdal_rasterize command line option
-3d, 564
--help, 563
--help-general, 563
--quiet, 563
-a, 564
-a_nodata, 565
-a_srs, 565
-add, 564
-at, 564
-b, 563
-burn, 564
-co, 565
-dialect, 564
-i, 563
-init, 565
-l, 564
-of, 564
-oo, 565
-optim, 565
-ot, 565
-q, 563
-sql, 564
-tap, 565
-te, 565
-to, 565
-tr, 565
-ts, 565
-where, 564
<dst_filename>, 566
<src_datasource>, 566
gdal_retile, 566
gdal_retile command line option
--help, 567
--help-general, 567
-co, 567
-csv, 568
-csvDelim, 568
-levels, 568
-of, 567
-ot, 568
-overlap, 568
-ps, 568
-pyramidOnly, 568
-r, 568
-resume, 568
-s_srs, 568
-targetDir, 567
-tileIndex, 568
-tileIndexField, 568
-useDirForEachRow, 568
-v, 568
gdal_sieve, 569
gdal_translate, 569
gdal_translate command line option
--help, 570
--help-general, 570
--quiet, 570
-a_coord_epoch, 574
-a_gt, 574
-a_nodata, 574
-a_offset, 574
-a_scale, 574
-a_srs, 573
-a_ullr, 574
-b, 571
-co, 575
-colorinterp, 574
-colorinterp_X, 574
-dmo, 575
-eco, 573
-epo, 573
-expand, 571

-exponent, 572
-gcp, 575
-if, 571
-mask, 571
-mo, 574
-nogcp, 575
-norat, 575
-noxmp, 576
-of, 571
-oo, 576
-ot, 570
-outsize, 571
-ovr, 571
-projwin, 573
-projwin_srs, 573
-q, 570, 575
-r, 572
-scale, 572
-sds, 575
-srcwin, 573
-stats, 575
-strict, 571
-tr, 571
-unscale, 572
<dst_dataset>, 576
<src_dataset>, 576
gdal_vector_buffer command line option
--active-geometry, 292
--active-layer, 292
--append, 292
--co, 292
--creation-option, 292
--distance, 289
--endcap-style, 289
--format, 293
--if, 292
--input-format, 292
--input-layer, 289
--join-style, 289
--layer-creation-option, 293
--lco, 293
--mitre-limit, 289
--of, 293
--oo, 293
--open-option, 293
--output-format, 293
--output-layer, 290
--output-oo, 293
--output-open-option, 293
--overwrite, 293
--overwrite-layer, 293
--quadrant-segments, 290
--side, 290
--skip-errors, 293
--update, 293
--upsert, 293
-f, 293
gdal_vector_check_coverage command line option
--append, 297
--co, 297
--creation-option, 297
--format, 298
--geometry-field, 297
--if, 297
--include-valid, 297
--input-format, 297
--input-layer, 297
--layer-creation-option, 297
--lco, 297
--maximum-gap-width, 297
--no-create-empty-layers, 298
--of, 298
--oo, 298
--open-option, 298
--output-format, 298
--output-layer, 297
--output-oo, 298
--output-open-option, 298
--overwrite, 298
--overwrite-layer, 298
--skip-errors, 298
--update, 298
--upsert, 298
-f, 298
gdal_vector_check_geometry command line option
--append, 302
--co, 302
--creation-option, 302
--format, 303
--geometry-field, 302
--if, 302
--include-field, 302
--include-valid, 302
--input-format, 302
--input-layer, 302
--layer-creation-option, 302
--lco, 302
--no-create-empty-layers, 303
--of, 303
--oo, 303
--open-option, 303
--output-format, 303
--output-layer, 302
--output-oo, 303
--output-open-option, 303
--overwrite, 303

--overwrite-layer, 303
--skip-errors, 303
--update, 303
--upsert, 303
-f, 303
gdal_vector_clean_coverage command line option
--active-layer, 308
--append, 308
--co, 308
--creation-option, 308
--format, 308
--if, 308
--input-format, 308
--input-layer, 306
--layer-creation-option, 308
--lco, 308
--maximum-gap-width, 306
--merge-strategy, 306
--of, 308
--oo, 308
--open-option, 308
--output-format, 308
--output-layer, 306
--output-oo, 308
--output-open-option, 308
--overwrite, 308
--overwrite-layer, 309
--skip-errors, 309
--snapping-distance, 306
--update, 309
--upsert, 309
-f, 308
gdal_vector_clip command line option
--active-layer, 312
--append, 312
--bbox, 311
--bbox-crs, 311
--co, 312
--creation-option, 312
--format, 313
--geometry, 311
--geometry-crs, 312
--if, 312
--input-format, 312
--input-layer, 312
--layer-creation-option, 313
--lco, 313
--like, 312
--like-layer, 312
--like-sql, 312
--like-where, 312
--of, 313
--oo, 313
--open-option, 313
--output-format, 313
--output-layer, 313
--output-oo, 313
--output-open-option, 313
--overwrite, 313
--overwrite-layer, 313
--skip-errors, 313
--update, 313
--upsert, 313
-f, 313
gdal_vector_combine command line option
--add-extra-fields, 316
--append, 317
--co, 317
--creation-option, 317
--format, 317
--group-by, 316
--if, 317
--input-format, 317
--keep-nested, 316
--layer-creation-option, 317
--lco, 317
--of, 317
--oo, 317
--open-option, 317
--output-format, 317
--output-oo, 317
--output-open-option, 317
--overwrite, 318
--overwrite-layer, 318
--skip-errors, 318
--update, 318
--upsert, 318
-f, 317
gdal_vector_concat command line option
--append, 321
--co, 321
--creation-option, 321
--field-strategy, 320
--format, 322
--if, 321
--input-crs, 320
--input-format, 321
--input-layer, 322
--layer-creation-option, 322
--lco, 322
--mode, 320
--of, 322
--oo, 322
--open-option, 322
--output-crs, 320
--output-format, 322
--output-layer, 321

--output-oo, 322
--output-open-option, 322
--overwrite, 322
--overwrite-layer, 322
--skip-errors, 322
--source-layer-field-content, 321
--source-layer-field-name, 321
--update, 322
--upsert, 322
-d, 320
-f, 322
-s, 320
gdal_vector_concave_hull command line option
--active-geometry, 325
--active-layer, 325
--allow-holes, 325
--append, 325
--co, 325
--creation-option, 325
--format, 326
--if, 325
--input-format, 325
--input-layer, 326
--layer-creation-option, 326
--lco, 326
--of, 326
--oo, 326
--open-option, 326
--output-format, 326
--output-layer, 326
--output-oo, 326
--output-open-option, 326
--overwrite, 326
--overwrite-layer, 326
--ratio, 325
--skip-errors, 326
--tight, 325
--update, 326
--upsert, 326
-f, 326
gdal_vector_convert command line option
--append, 329
--co, 329
--creation-option, 329
--format, 330
--if, 329
--input-format, 329
--input-layer, 330
--layer-creation-option, 330
--lco, 330
--no-create-empty-layers, 330
--of, 330
--oo, 330
--open-option, 330
--output-format, 330
--output-layer, 330
--output-oo, 330
--output-open-option, 330
--overwrite, 330
--overwrite-layer, 330
--quiet, 331
--skip-errors, 330
--update, 330
--upsert, 331
-f, 330
-q, 331
gdal_vector_convex_hull command line option
--active-geometry, 333
--active-layer, 333
--append, 333
--co, 333
--creation-option, 333
--format, 334
--if, 333
--input-format, 333
--input-layer, 333
--layer-creation-option, 334
--lco, 334
--of, 334
--oo, 334
--open-option, 334
--output-format, 334
--output-layer, 334
--output-oo, 334
--output-open-option, 334
--overwrite, 334
--overwrite-layer, 334
--skip-errors, 334
--update, 334
--upsert, 334
-f, 334
gdal_vector_create command line option
--crs, 337
--fid, 338
--field, 337
--geometry-field, 337
--geometry-type, 337
--input, 337
--input-layer, 337
--layer, 337
--like, 337
--oo, 338
--open-option, 338
--output-layer, 337
--schema, 337
-i, 337
gdal_vector_dissolve command line option

--active-geometry, 340
--active-layer, 340
--append, 340
--co, 340
--creation-option, 340
--format, 341
--if, 340
--input-format, 340
--layer-creation-option, 341
--lco, 341
--of, 341
--oo, 341
--open-option, 341
--output-format, 341
--output-oo, 341
--output-open-option, 341
--overwrite, 341
--overwrite-layer, 341
--skip-errors, 341
--update, 341
--upsert, 341
-f, 341
gdal_vector_edit command line option
--active-layer, 344
--append, 344
--co, 344
--creation-option, 344
--crs, 344
--format, 345
--geometry-type, 344
--if, 345
--input-format, 345
--input-layer, 345
--layer-creation-option, 345
--layer-metadata, 344
--lco, 345
--metadata, 344
--of, 345
--oo, 345
--open-option, 345
--output-format, 345
--output-layer, 345
--output-oo, 345
--output-open-option, 345
--overwrite, 345
--overwrite-layer, 345
--skip-errors, 345
--unset-fid, 344
--unset-layer-metadata, 344
--unset-metadata, 344
--update, 346
--upsert, 346
-f, 345
gdal_vector_explode_collections command line option
--active-geometry, 365
--active-layer, 365
--append, 365
--co, 365
--creation-option, 365
--format, 366
--geometry-type, 365
--if, 365
--input-format, 365
--input-layer, 365
--layer-creation-option, 366
--lco, 366
--of, 366
--oo, 366
--open-option, 366
--output-format, 366
--output-layer, 366
--output-oo, 366
--output-open-option, 366
--overwrite, 366
--overwrite-layer, 366
--skip-errors, 366
--skip-on-type-mismatch, 365
--update, 366
--upsert, 366
-f, 366
gdal_vector_export_schema command line option
--if, 362
--input-format, 362
--input-layer, 361
--layer, 361
--oo, 362
--open-option, 362
--output, 362
--overwrite, 362
-l, 361
-o, 362
gdal_vector_filter command line option
--active-layer, 348
--append, 348
--bbox, 348
--co, 348
--creation-option, 348
--format, 349
--if, 348
--input-format, 348
--input-layer, 349
--layer-creation-option, 349
--lco, 349
--of, 349
--oo, 349

--open-option, 349
--output-format, 349
--output-layer, 349
--output-oo, 349
--output-open-option, 349
--overwrite, 349
--overwrite-layer, 349
--skip-errors, 349
--update, 349
--update-extent, 348
--upsert, 350
--where, 348
-f, 349
gdal_vector_grid command line option
--angle, 370–373
--bbox, 369
--co, 368
--creation-option, 368
--crs, 369
--datatype, 369
--extent, 368
--format, 368
--if, 369
--input-format, 369
--layer, 368
--layer-name, 368
--max-points, 370, 371
--max-points-per-quadrant, 370, 371, 373
--min-points, 370, 371, 373
--min-points-per-quadrant, 370–373
--nodata, 370–373
--of, 368
--oo, 369
--open-option, 369
--ot, 369
--output-data-type, 369
--output-format, 368
--overwrite, 368
--power, 369, 370
--radius, 369–373
--radius1, 370–373
--radius2, 370–373
--resolution, 368
--size, 368
--smoothing, 369, 370
--sql, 368
--zfield, 369
--zincrease, 369
--zmultiply, 369
-f, 368
-l, 368
gdal_vector_index command line option
--absolute-path, 377
--accept-different-crs, 377
--accept-different-schemas, 377
--append, 378
--co, 378
--creation-option, 378
--dataset-name-only, 377
--filename-filter, 377
--format, 378
--layer-creation-option, 378
--lco, 378
--location-name, 377
--metadata, 377
--nln, 378
--of, 378
--output-crs, 377
--output-format, 378
--output-layer, 378
--overwrite, 378
--overwrite-layer, 378
--recursive, 377
--source-crs-field-name, 377
--source-crs-format, 377
--source-layer-index, 377
--source-layer-name, 377
--update, 378
-f, 378
gdal_vector_info command line option
--crs-format, 353
--dialect, 352
--features, 352
--fid, 352
--format, 352
--if, 353
--input-format, 353
--input-layer, 352
--layer, 352
--limit, 352
--of, 352
--oo, 353
--open-option, 353
--output-format, 352
--sql, 352
--summary, 352
--where, 352
-f, 352
-l, 352
gdal_vector_layer_algebra command line option
--all-input-field, 383
--all-method-field, 383
--append, 383
--co, 383
--creation-option, 383
--format, 384
--geometry-type, 380

--if, 384
--input-field, 383
--input-format, 384
--input-layer, 380
--input-prefix, 383
--layer-creation-option, 384
--lco, 384
--method-field, 383
--method-layer, 383
--method-prefix, 383
--no-input-field, 383
--no-method-field, 383
--of, 384
--oo, 384
--open-option, 384
--operation, 381
--output-format, 384
--output-layer, 384
--overwrite, 384
--overwrite-layer, 384
--update, 384
-f, 384
gdal_vector_make_point command line option
--append, 386
--co, 386
--creation-option, 386
--format, 387
--if, 387
--input-format, 387
--input-layer, 387
--layer-creation-option, 387
--lco, 387
--m, 386
--of, 387
--oo, 387
--open-option, 387
--output-crs, 386
--output-format, 387
--output-layer, 387
--output-oo, 387
--output-open-option, 387
--overwrite, 387
--overwrite-layer, 387
--skip-errors, 388
--update, 388
--upsert, 388
--x, 386
--y, 386
--z, 386
-f, 387
gdal_vector_make_valid command line option
--active-geometry, 390
--active-layer, 390
--append, 390
--co, 390
--creation-option, 390
--format, 391
--if, 391
--input-format, 391
--input-layer, 391
--keep-lower-dim, 390
--layer-creation-option, 391
--lco, 391
--method, 390
--of, 391
--oo, 391
--open-option, 391
--output-format, 391
--output-layer, 391
--output-oo, 391
--output-open-option, 391
--overwrite, 391
--overwrite-layer, 391
--skip-errors, 391
--update, 391
--upsert, 392
-f, 391
gdal_vector_materialize command line option
--co, 393
--creation-option, 393
--layer-creation-option, 393
--lco, 393
--output, 393
--overwrite, 393
gdal_vector_partition command line option
--append, 396
--co, 396
--creation-option, 396
--feature-limit, 395
--field, 395
--format, 397
--if, 396
--input-format, 396
--layer-creation-option, 397
--lco, 397
--max-file-size, 395
--of, 397
--omit-partitioned-field, 396
--oo, 397
--open-option, 397
--output, 396
--output-format, 397
--overwrite, 397
--pattern, 396
--scheme, 396
--skip-errors, 397
-f, 397
gdal_vector_rasterize command line option

--3d, 423
--add, 423
--all-touched, 423
--attribute-name, 423
--band, 423
--burn, 423
--co, 425
--creation-option, 425
--crs, 423
--datatype, 424
--dialect, 424
--extent, 423
--format, 425
--if, 425
--init, 423
--input-format, 425
--input-layer, 423
--invert, 423
--nodata, 424
--of, 425
--oo, 425
--open-option, 425
--optimization, 424
--ot, 424
--output-data-type, 424
--output-format, 425
--overwrite, 425
--resolution, 424
--size, 424
--sql, 424
--tap, 424
--target-aligned-pixels, 424
--transformer-option, 425
--update, 425
--where, 425
-a, 423
-b, 423
-f, 425
-l, 423
gdal_vector_read command line option
--if, 427
--input-format, 427
--oo, 427
--open-option, 427
gdal_vector_rename_layer command line option
--append, 430
--ascii, 429
--co, 430
--creation-option, 430
--filename-compatible, 429
--format, 430
--if, 430
--input-format, 430
--input-layer, 429
--layer-creation-option, 430
--lco, 430
--lower-case, 429
--max-length, 430
--of, 430
--oo, 430
--open-option, 430
--output-format, 430
--output-layer, 429, 430
--output-oo, 430
--output-open-option, 430
--overwrite, 431
--overwrite-layer, 431
--replacement-character, 429
--reserved-characters, 429
--skip-errors, 431
--update, 431
--upsert, 431
-f, 430
gdal_vector_reproject command line option
--active-layer, 434
--append, 434
--co, 434
--creation-option, 434
--format, 434
--if, 434
--input-crs, 433
--input-format, 434
--input-layer, 434
--layer-creation-option, 434
--lco, 434
--of, 434
--oo, 434
--open-option, 434
--output-crs, 433
--output-format, 434
--output-layer, 434
--output-oo, 435
--output-open-option, 435
--overwrite, 435
--overwrite-layer, 435
--quiet, 435
--skip-errors, 435
--update, 435
--upsert, 435
-d, 433
-f, 434
-q, 435
-s, 433
<INPUT-CRS>, 433
gdal_vector_segmentize command line option
--active-geometry, 442
--active-layer, 442

--append, 442
--co, 442
--creation-option, 442
--format, 443
--if, 443
--input-format, 443
--input-layer, 443
--layer-creation-option, 443
--lco, 443
--max-length, 442
--of, 443
--oo, 443
--open-option, 443
--output-format, 443
--output-layer, 443
--output-oo, 443
--output-open-option, 443
--overwrite, 443
--overwrite-layer, 444
--skip-errors, 444
--update, 444
--upsert, 444
-f, 443
gdal_vector_select command line option
--active-layer, 439
--append, 439
--co, 439
--creation-option, 439
--exclude, 438
--fields, 438
--format, 439
--if, 439
--ignore-missing-fields, 438
--input-format, 439
--input-layer, 439
--layer-creation-option, 439
--lco, 439
--of, 439
--oo, 439
--open-option, 439
--output-format, 439
--output-layer, 439
--output-oo, 440
--output-open-option, 440
--overwrite, 440
--overwrite-layer, 440
--skip-errors, 440
--update, 440
--upsert, 440
-f, 439
gdal_vector_set_field_type command line option
--active-layer, 446
--append, 446
--co, 446
--creation-option, 446
--field-name, 446
--field-type, 446
--format, 447
--if, 447
--input-field-type, 446
--input-format, 447
--input-layer, 447
--layer-creation-option, 447
--lco, 447
--of, 447
--oo, 447
--open-option, 447
--output-field-type, 446
--output-format, 447
--output-layer, 447
--output-oo, 447
--output-open-option, 447
--overwrite, 447
--overwrite-layer, 447
--skip-errors, 447
--update, 448
--upsert, 448
-f, 447
gdal_vector_set_geom_type command line option
--active-geometry, 451
--active-layer, 451
--append, 451
--co, 451
--creation-option, 451
--curve, 450
--dim, 450
--feature-only, 450
--format, 452
--geometry-type, 450
--if, 452
--input-format, 452
--input-layer, 452
--layer-creation-option, 452
--layer-only, 451
--lco, 452
--linear, 451
--multi, 451
--of, 452
--oo, 452
--open-option, 452
--output-format, 452
--output-layer, 452
--output-oo, 452
--output-open-option, 452
--overwrite, 452
--overwrite-layer, 452

--single, 451
--skip, 451
--skip-errors, 452
--update, 453
--upsert, 453
-f, 452
gdal_vector_simplify command line option
--active-geometry, 458
--active-layer, 458
--append, 458
--co, 458
--creation-option, 458
--format, 458
--if, 458
--input-format, 458
--input-layer, 458
--layer-creation-option, 458
--lco, 458
--of, 458
--oo, 458
--open-option, 458
--output-format, 458
--output-layer, 459
--output-oo, 459
--output-open-option, 459
--overwrite, 459
--overwrite-layer, 459
--skip-errors, 459
--tolerance, 457
--update, 459
--upsert, 459
-f, 458
gdal_vector_simplify_coverage command line option
--active-layer, 462
--append, 462
--co, 462
--creation-option, 462
--format, 463
--if, 462
--input-format, 462
--input-layer, 462
--layer-creation-option, 462
--lco, 462
--of, 463
--oo, 463
--open-option, 463
--output-format, 463
--output-layer, 463
--output-oo, 463
--output-open-option, 463
--overwrite, 463
--overwrite-layer, 463
--preserve-boundary, 462
--skip-errors, 463
--tolerance, 462
--update, 463
--upsert, 463
-f, 463
gdal_vector_sort command line option
--geometry-field, 455
--method, 455
--use-tempfile, 455
gdal_vector_sql command line option
--append, 466
--co, 466
--creation-option, 466
--dialect, 466
--format, 467
--if, 466
--input-format, 466
--layer-creation-option, 467
--lco, 467
--of, 467
--oo, 467
--open-option, 467
--output-format, 467
--output-layer, 465
--output-oo, 467
--output-open-option, 467
--overwrite, 467
--overwrite-layer, 467
--quiet, 466, 468
--skip-errors, 467
--sql, 466
--update, 467
--upsert, 467
-f, 467
-q, 468
gdal_vector_swap_xy command line option
--active-geometry, 470
--active-layer, 470
--append, 470
--co, 470
--creation-option, 470
--format, 471
--if, 471
--input-format, 471
--input-layer, 471
--layer-creation-option, 471
--lco, 471
--of, 471
--oo, 471
--open-option, 471
--output-format, 471
--output-layer, 471
--output-oo, 471
--output-open-option, 471

--overwrite, 471
--overwrite-layer, 471
--skip-errors, 471
--update, 472
--upsert, 472
-f, 471
gdal_vector_update command line option
--if, 474
--input-format, 474
--input-layer, 473
--key, 473
--mode, 473
--oo, 474
--open-option, 474
--output-layer, 474
--output-oo, 474
--output-open-option, 474
gdal_vector_write command line option
--append, 476
--co, 475
--creation-option, 475
--format, 475
--layer-creation-option, 475
--lco, 475
--of, 475
--output-format, 475
--output-layer, 476
--output-oo, 476
--output-open-option, 476
--overwrite, 476
--overwrite-layer, 476
--skip-errors, 476
--update, 476
--upsert, 476
-f, 475
gdal_viewshed, 577
gdal_viewshed command line option
--help, 578
--help-general, 578
--quiet, 578
-a_nodata, 578
-b, 578
-cc, 578
-co, 578
-iv, 579
-j, 580
-md, 578
-om, 579
-os, 579
-ov, 579
-ox, 578
-oy, 578
-oz, 578
-q, 578
-tz, 578
-vv, 579
gdal_vsi_copy command line option
--recursive, 496
--skip-errors, 496
-r, 496
gdal_vsi_delete command line option
-R, 497
--recursive, 497
-r, 497
gdal_vsi_list command line option
-R, 499
--abs, 499
--absolute-path, 499
--depth, 499
--filename, 499
--format, 499
--long, 499
--long-listing, 499
--of, 499
--output-format, 499
--recursive, 499
--tree, 499
-f, 499
-l, 499
gdal_vsi_sozip command line option
--content-type, 504
--enable-sozip, 504, 505
--input, 503, 505–507
--junk-paths, 503
--no-paths, 503
--output, 503, 505
--overwrite, 503, 505
--quiet, 504, 505, 507
--recursive, 503
--sozip-chunk-size, 504, 505
--sozip-min-file-size, 504, 505
--verbose, 507
-i, 503, 505–507
-j, 503
-o, 503, 505
-q, 504, 505, 507
-r, 503
-v, 507
gdal_vsi_sync command line option
--num-threads, 501
--recursive, 501
--strategy, 501
-j, 501
-r, 501
gdal-config, 522
gdaladdo, 580
gdaladdo command line option
--help, 580

--help-general, 582
--partial-refresh-from-projwin, 583
--partial-refresh-from-source-extent, 583
--partial-refresh-from-source-timestamp, 582
--quiet, 582
-b, 582
-clean, 582
-minsize, 582
-oo, 582
-q, 582
-r, 582
-ro, 582
<filename>, 583
<levels>, 583
gdalattachpct, 587
gdalattachpct command line option
--help, 587
--help-general, 587
-of, 587
<dest_file>, 587
<palette_file>, 587
<source_file>, 587
gdalbuildvrt, 587
gdalbuildvrt command line option
--help, 589
--help-general, 589
--quiet, 589
-a_srs, 591
-addalpha, 589
-allow_projection_difference, 590
-b, 590
-co, 591
-hidenodata, 589
-ignore_srcmaskband, 590
-input_file_list, 591
-nodata_max_mask_threshold, 590
-non_strict, 591
-oo, 591
-overwrite, 591
-pixel-function, 590
-pixel-function-arg, 590
-q, 589, 591
-r, 591
-resolution, 589
-sd, 590
-separate, 590
-srcnodata, 589
-strict, 591
-tap, 589
-te, 589
-tileindex, 589
-tr, 589
-vrtnodata, 590
-write_absolute_path, 591
gdalcompare, 592
gdalcompare command line option
--help, 592
--help-general, 592
-dumpdiffs, 592
-sds, 593
-skip_binary, 592
-skip_geolocation, 593
-skip_geotransform, 593
-skip_metadata, 593
-skip_overviews, 593
-skip_rpc, 593
-skip_srs, 593
<golden_file>, 593
<new_file>, 593
gdaldem, 594
gdalenhance, 602
gdalenhance command line option
--help, 602
--help-general, 602
-co, 602
-config, 603
-equalize, 603
-of, 602
-ot, 603
gdalinfo, 604
gdalinfo command line option
--help, 605
--help-general, 605
-approx_stats, 605
-checksum, 606
-hist, 605
-if, 606
-json, 605
-listmdd, 606
-mdd, 606
-mm, 605
-noct, 605
-nofl, 606
-nogcp, 605
-nomask, 606
-nomd, 605
-nonodata, 605
-norat, 605
-oo, 606
-proj4, 606
-sd, 606
-stats, 605
-wkt_format, 606
gdallocationinfo, 614
gdallocationinfo command line option
-E, 615
--help, 614

--help-general, 614
-b, 614
-field_sep, 615
-geoloc, 615
-ignore_extra_input, 615
-l_srs, 615
-lifonly, 614
-oo, 615
-overview, 615
-r, 614
-valonly, 614
-wgs84, 615
-xml, 614
<srcfile>, 615
<x>, 615
<y>, 615
gdalmanage, 617
gdalmdiminfo, 649
gdalmdiminfo command line option
--help, 649
--help-general, 649
-array, 649
-arrayoption, 649
-detailed, 649
-if, 649
-limit, 649
-nopretty, 649
-oo, 649
-stats, 649
gdalmdimtranslate, 652
gdalmdimtranslate command line option
--help, 652
--help-general, 653
--quiet, 653
-array, 653
-arrayoption, 653
-co, 653
-group, 653
-if, 653
-of, 653
-oo, 654
-q, 653
-scaleaxes, 654
-strict, 654
-subset, 654
<dst_dataset>, 654
<src_dataset>, 654
gdalmove, 619
gdalsrsinfo, 620
gdalsrsinfo command line option
-V, 620
--help, 620
--help-general, 620
--single-line, 620
-e, 620
-o, 620
<srs_def>, 621
gdaltindex, 624
gdaltindex command line option
--help, 625
--help-general, 625
--quiet, 625
-bandcount, 627
-colorinterp, 627
-fetch_md, 628
-filename_filter, 625
-gti_filename, 627
-lco, 626
-lyr_name, 626
-mask, 628
-max_pixel_size, 625
-min_pixel_size, 625
-mo, 628
-nodata, 627
-of, 625
-ot, 627
-overwrite, 625
-q, 625
-recursive, 625
-skip_different_projection, 626
-src_srs_format, 626
-src_srs_name, 626
-t_srs, 626
-te, 627
-tileindex, 626
-tr, 627
-write_absolute_path, 626
<file_or_dir>, 626
<index_file>, 626
gdaltransform, 629
gdaltransform command line option
-E, 631
--help, 629
--help-general, 629
-ct, 630
-field_sep, 631
-gcp, 630
-geoloc, 630
-i, 630
-ignore_extra_input, 630
-order, 630
-output_xy, 630
-rpc, 630
-s_coord_epoch, 630
-s_srs, 629
-t_coord_epoch, 630
-t_srs, 630
-to, 630

-tps, 630
<dstfile>, 631
<srcfile>, 631
gdalwarp, 632
gdalwarp command line option
--help, 633
--help-general, 633
--quiet, 633
-b, 633
-cblend, 639
-cl, 639
-co, 638
-crop_to_cutline, 639
-csql, 639
-ct, 635
-cutline, 639
-cutline_srs, 639
-cvmd, 639
-cwhere, 639
-doo, 639
-dstalpha, 638
-dstband, 634
-dstnodata, 638
-et, 636
-geoloc, 636
-if, 638
-multi, 638
-nomd, 639
-nosrcalpha, 638
-novshift, 635
-of, 638
-oo, 639
-order, 635
-ot, 637
-overwrite, 639
-ovr, 636
-q, 633
-r, 637
-refine_gcps, 636
-rpc, 636
-s_coord_epoch, 634
-s_srs, 634
-setci, 639
-srcalpha, 638
-srcband, 633
-srcnodata, 637
-t_coord_epoch, 635
-t_srs, 634
-tap, 636
-te, 636
-te_srs, 636
-to, 635
-tps, 635
-tr, 636
-ts, 636
-vshift, 635
-wm, 638
-wo, 637
-wt, 637
<dst_dataset_name>, 640
<src_dataset_name>, 639
gnmanalyse, 688
gnmanalyse command line option
--help, 688
--help-general, 688
-alo, 689
-d, 688
-dsco, 689
-f, 689
-l, 689
-lco, 689
<gnm_name>, 689
dijkstra, 688
kpaths, 688
resource, 688
gnmmanage, 686
gnmmanage command line option
--help, 686
--help-general, 687
-bl, 687
-c, 687
-dir, 687
-dsco, 687
-f, 687
-ic, 687
-info, 687
-l, 687
-t_srs, 687
-unbl, 688
-unblall, 688
<gnm_name>, 688
<layer>, 688
autoconnect, 687
change, 687
connect, 687
create, 687
delete, 687
disconnect, 687
import, 687
rule, 687

I

import
gnmmanage command line option, 687

K

kpaths
gnmanalyse command line option, 688

N

nearblack, 644
nearblack command line option
--help, 645
--help-general, 645
--quiet, 645
-alg, 646
-co, 645
-color, 645
-nb, 645
-near, 645
-o, 645
-of, 645
-q, 645
-setalpha, 645
-setmask, 645
-white, 645
<infile>, 646

O

ogr2ogr, 665
ogr2ogr command line option
--help, 667
--help-general, 667
--quiet, 667
-a_coord_epoch, 669
-a_srs, 669
-addfields, 674
-append, 667
-clipdst, 672
-clipdstlayer, 672
-clipdstsql, 672
-clipdstwhere, 672
-clipsrc, 672
-clipsrclayer, 672
-clipsrcsql, 672
-clipsrcwhere, 672
-ct, 671
-ct_opt, 671
-dateTimeTo, 673
-datelineoffset, 672
-dialect, 668
-dim, 669
-doo, 671
-ds_transaction, 672
-dsco, 669
-emptyStrAsNull, 674
-explodecollections, 674
-f, 667
-fid, 671
-fieldTypeToString, 673
-fieldmap, 674
-forceNullable, 674
-gcp, 674
-geomfield, 668
-gt, 672
-if, 667
-lco, 669
-limit, 671
-mRes, 670
-makevalid, 673
-mapFieldType, 673
-maxsubfields, 674
-mo, 675
-nln, 669
-nlt, 669
-noNativeData, 675
-nomd, 675
-of, 667
-oo, 671
-order, 674
-overwrite, 667
-preserve_fid, 671
-progress, 668
-q, 667
-relaxedFieldNameMatch, 674
-resolveDomains, 675
-s_coord_epoch, 670
-s_srs, 670
-segmentize, 673
-select, 668
-simplify, 672
-skipfailures, 668
-skipinvalid, 673
-spat, 668
-spat_srs, 668
-splitlistfields, 673
-sql, 668
-t_coord_epoch, 670
-t_srs, 669
-tps, 674
-unsetCoordPrecision, 670
-unsetDefault, 674
-unsetFid, 674
-unsetFieldWidth, 673
-update, 668
-upsert, 667
-where, 668
-wrapdateline, 672
-xyRes, 670
-zRes, 670
-zfield, 674
<dst_dataset_name>, 675
<layer_name>, 675
<src_dataset_name>, 675
ogr_layer_algebra, 684
ogr_layer_algebra command line option
--help, 684

--help-general, 684
-a_srs, 686
-dsco, 685
-f, 685
-input_ds, 685
-input_fields, 686
-input_lyr, 685
-lco, 685
-method_ds, 685
-method_fields, 686
-method_lyr, 685
-nlt, 686
-opt, 685
-output_ds, 685
-output_lyr_name, 685
-overwrite, 685
<mode>, 684
ogrinfo, 656
ogrinfo command line option
--formats, 659
--help, 657
--help-general, 657
-al, 657
-dialect, 658
-extent3D, 659
-features, 657
-fid, 658
-fielddomain, 658
-fields, 658
-geom, 658
-geomfield, 658
-if, 657
-json, 657
-limit, 657
-listmdd, 659
-mdd, 659
-nocount, 659
-noextent, 659
-nogeomtype, 659
-nomd, 659
-oo, 658
-q, 657
-rl, 657
-ro, 657
-so, 657
-spat, 658
-sql, 658
-where, 658
-wkt_format, 659
<datasource_name>, 659
<layer>, 660
ogrlineref, 678
ogrmerge, 681
ogrmerge command line option
--help, 681
--help-general, 682
-a_srs, 683
-append, 682
-dsco, 682
-f, 682
-field_strategy, 683
-lco, 682
-nln, 682
-o, 682
-overwrite_ds, 682
-overwrite_layer, 682
-progress, 683
-s_srs, 683
-single, 682
-skipfailures, 683
-src_geom_type, 682
-src_layer_field_content, 683
-src_layer_field_name, 683
-t_srs, 683
-update, 682
<src_dsname>, 682
ogrtindex, 677
ogrtindex command line option
--help, 677
--help-general, 677
-accept_different_schemas, 678
-f, 677
-lname, 677
-lnum, 677
-skip_different_projection, 678
-src_srs_format, 678
-src_srs_name, 678
-t_srs, 678
-tileindex, 678
-write_absolute_path, 678

P

pct2rgb, 646
pct2rgb command line option
--help, 646
--help-general, 646
-b, 647
-of, 647
-pct, 647
-rgba, 647
<dest_file>, 647
<source_file>, 647

R

resource
gnmanalyse command line option, 688
return_ds
command line option, 532

rgb2pct, 647
rgb2pct command line option
    --creation-option, 648
    --help, 647
    --help-general, 648
    -n, 648
    -of, 648
    -pct, 648
    <dest_file>, 648
    <source_file>, 648
rule
    gnmmanage command line option, 687
Run()
    built-in function, 31

S

sozip, 689
sozip command line option
    --content-type, 691
    --enable-sozip, 690
    --grow, 690
    --help, 690
    --help-general, 690
    --junk-paths, 690
    --list, 690
    --optimize-from, 690
    --overwrite, 690
    --quiet, 690
    --recurse-paths, 690
    --sozip-chunk-size, 691
    --sozip-min-file-size, 691
    --validate, 690
    --verbose, 690
    -g, 690
    -j, 690
    -l, 690
    -r, 690
    <filename>, 691
    <zip_filename>, 691

T

TestFunction (*C++ function*), 1096

U

user_namespace
    command line option, 532

V

vector_common_options command line option
    --build, 655
    --config, 656
    --debug, 656
    --format, 656
    --formats, 655, 656
    --help-general, 656
    --optfile, 656
    --version, 655