



GDAL

Frank Warmerdam, Even Rouault, and others

Apr 27, 2020

CONTENTS

1	What is GDAL?	1
2	Download	3
2.1	Current Releases	3
2.2	Past Releases	3
2.3	Development Source	3
2.4	Binaries	3
2.4.1	Windows	4
2.4.2	Debian	4
2.4.3	Conda	4
2.4.4	Linux Docker images	4
3	Programs	5
3.1	Raster programs	5
3.1.1	Common options for raster programs	5
3.1.1.1	Creating new files	6
3.1.2	gdalinfo	7
3.1.2.1	Synopsis	7
3.1.2.2	Description	7
3.1.2.3	C API	9
3.1.2.4	Example	9
3.1.3	gdal_translate	9
3.1.3.1	Synopsis	9
3.1.3.2	Description	10
3.1.3.3	C API	13
3.1.3.4	Examples	13
3.1.4	gdaladdo	13
3.1.4.1	Synopsis	13
3.1.4.2	Description	13
3.1.4.3	External overviews in GeoTIFF format	14
3.1.4.4	C API	15
3.1.4.5	Examples	15
3.1.5	gdalwarp	16
3.1.5.1	Synopsis	16
3.1.5.2	Description	16
3.1.5.3	Examples	20
3.1.5.4	See also	21
3.1.6	gdaltindex	21
3.1.6.1	Synopsis	21
3.1.6.2	Description	21

3.1.6.3	Examples	22
3.1.6.4	See also	22
3.1.7	gdalbuildvrt	22
3.1.7.1	Synopsis	22
3.1.7.2	Description	22
3.1.7.3	Examples	24
3.1.8	gdal_contour	25
3.1.8.1	Synopsis	25
3.1.8.2	Description	25
3.1.8.3	C API	26
3.1.8.4	Example	26
3.1.9	gdaldem	26
3.1.9.1	Synopsis	26
3.1.9.2	Description	27
3.1.9.3	Modes	29
3.1.9.4	C API	32
3.1.9.5	Authors	32
3.1.9.6	See also	32
3.1.10	rgb2pct	32
3.1.10.1	Synopsis	32
3.1.10.2	Description	32
3.1.10.3	Example	33
3.1.11	pct2rgb	33
3.1.11.1	Synopsis	33
3.1.11.2	Description	33
3.1.12	gdal_merge	34
3.1.12.1	Synopsis	34
3.1.12.2	Description	34
3.1.12.3	Example	35
3.1.13	gdal2tiles	35
3.1.13.1	Synopsis	35
3.1.13.2	Description	35
3.1.13.3	Examples	37
3.1.14	gdal_rasterize	37
3.1.14.1	Synopsis	37
3.1.14.2	Description	37
3.1.14.3	C API	40
3.1.14.4	Example	40
3.1.15	gdaltransform	40
3.1.15.1	Synopsis	40
3.1.15.2	Description	40
3.1.15.3	Examples	41
3.1.16	nearblack	42
3.1.16.1	Synopsis	42
3.1.16.2	Description	42
3.1.16.3	C API	43
3.1.17	gdal_retile	44
3.1.17.1	Synopsis	44
3.1.17.2	Description	44
3.1.18	gdal_grid	45
3.1.18.1	Synopsis	45
3.1.18.2	Description	46
3.1.18.3	Interpolation algorithms	47
3.1.18.4	Data metrics	49

3.1.18.5	Reading comma separated values	50
3.1.18.6	C API	50
3.1.18.7	Examples	51
3.1.19	gdal_proximity	51
3.1.19.1	Synopsis	51
3.1.19.2	Description	51
3.1.20	gdal_polygonize	52
3.1.20.1	Synopsis	52
3.1.20.2	Description	52
3.1.21	gdal_sieve	53
3.1.21.1	Synopsis	53
3.1.21.2	Description	53
3.1.22	gdal_fillnodata	54
3.1.22.1	Synopsis	54
3.1.22.2	Description	54
3.1.23	gdallocationinfo	54
3.1.23.1	Synopsis	54
3.1.23.2	Description	55
3.1.23.3	Examples	56
3.1.24	gdalsrsinfo	57
3.1.24.1	Synopsis	57
3.1.24.2	Description	57
3.1.24.3	Example	58
3.1.25	gdalmove	62
3.1.25.1	Synopsis	62
3.1.25.2	Description	62
3.1.26	gdal_edit	62
3.1.26.1	Synopsis	62
3.1.26.2	Description	63
3.1.26.3	Example	65
3.1.27	gdal_calc.py	65
3.1.27.1	Synopsis	65
3.1.27.2	Example	66
3.1.28	gdal_pansharpen.py	66
3.1.28.1	Synopsis	66
3.1.28.2	Description	66
3.1.28.3	Example	67
3.1.29	gdal-config	68
3.1.29.1	Synopsis	68
3.1.29.2	Description	68
3.1.30	gdalmanage	69
3.1.30.1	Synopsis	69
3.1.30.2	Description	69
3.1.30.3	Examples	69
3.1.31	gdalcompare	70
3.1.31.1	Synopsis	70
3.1.31.2	Description	70
3.1.32	gdal_viewshed	71
3.1.32.1	Synopsis	71
3.1.32.2	Description	71
3.1.32.3	C API	72
3.1.32.4	Example	72
3.2	Multidimensional Raster programs	75
3.2.1	gdalmdiminfo	75

3.2.1.1	Synopsis	75
3.2.1.2	Description	75
3.2.1.3	C API	75
3.2.1.4	Examples	75
3.2.2	gdalmdimtranslate	78
3.2.2.1	Synopsis	78
3.2.2.2	Description	78
3.2.2.3	C API	79
3.2.2.4	Examples	79
3.3	Vector programs	80
3.3.1	Common options for vector programs	80
3.3.2	ogrinfo	80
3.3.2.1	Synopsis	80
3.3.2.2	Description	81
3.3.2.3	Example	82
3.3.3	ogr2ogr	84
3.3.3.1	Synopsis	84
3.3.3.2	Description	85
3.3.3.3	Performance Hints	89
3.3.3.4	C API	90
3.3.3.5	Examples	90
3.3.4	ogrindex	91
3.3.4.1	Synopsis	91
3.3.4.2	Description	91
3.3.4.3	Example	92
3.3.5	ogrlineref	92
3.3.5.1	Synopsis	92
3.3.5.2	Description	92
3.3.5.3	Example	94
3.3.6	ogrmerge	94
3.3.6.1	Synopsis	94
3.3.6.2	Description	94
3.3.6.3	Examples	96
3.4	Geographic network programs	97
3.4.1	gnmmanage	97
3.4.1.1	Synopsis	97
3.4.1.2	Description	97
3.4.2	gnmanalyse	98
3.4.2.1	Synopsis	98
3.4.2.2	Description	98
4	Raster drivers	101
4.1	AAIGrid – Arc/Info ASCII Grid	105
4.1.1	Driver capabilities	106
4.2	ACE2 – ACE2	106
4.2.1	Driver capabilities	107
4.3	ADRG – ADRG/ARC Digitized Raster Graphics (.gen/.thf)	107
4.3.1	Driver capabilities	108
4.4	AIG – Arc/Info Binary Grid	108
4.4.1	Driver capabilities	109
4.4.1.1	Arc/Info Binary Grid Format	109
4.5	AIRSAR – AIRSAR Polarimetric Format	120
4.5.1	Driver capabilities	121
4.5.2	See Also	121

4.6	ARG – Azavea Raster Grid	121
4.6.1	Driver capabilities	121
4.7	BAG – Bathymetry Attributed Grid	122
4.7.1	Driver capabilities	122
4.7.2	Variable resolution (VR) grid support	123
4.7.3	Creation support	124
4.7.4	Usage examples	125
4.7.5	See Also	127
4.8	BLX – Magellan BLX Topo File Format	127
4.8.1	Driver capabilities	127
4.8.2	Georeferencing	128
4.8.3	Creation Issues	128
4.9	BMP – Microsoft Windows Device Independent Bitmap	128
4.9.1	Driver capabilities	129
4.9.2	Creation Options	129
4.9.3	See Also	129
4.10	BPG – Better Portable Graphics	129
4.11	BSB – Maptech/NOAA BSB Nautical Chart Format	130
4.11.1	Driver capabilities	130
4.11.2	Metadata	130
4.12	BT – VTP .bt Binary Terrain Format	131
4.12.1	Driver capabilities	131
4.13	BYN - Natural Resources Canada's Geoid file format (.byn)	132
4.13.1	Driver capabilities	132
4.13.2	Factor	133
4.13.3	See Also	133
4.14	CAD – AutoCAD DWG raster layer	133
4.14.1	Driver capabilities	134
4.15	CALS – CALS Type 1	134
4.15.1	Driver capabilities	134
4.15.2	Metadata	135
4.15.3	Creation issues	135
4.15.4	See Also	135
4.16	CEOS – CEOS Image	135
4.16.1	Driver capabilities	136
4.17	COASP – DRDC COASP SAR Processor Raster	136
4.18	COG – Cloud Optimized GeoTIFF generator	136
4.18.1	Driver capabilities	137
4.18.2	Creation Options	137
4.18.2.1	General creation options	137
4.18.2.2	Reprojection related creation options	139
4.18.3	File format details	139
4.18.3.1	High level	139
4.18.3.2	Low level	140
4.18.3.3	Header ghost area	140
4.18.3.4	Tile data leader and trailer	141
4.18.4	Examples	142
4.18.5	See Also	142
4.19	COSAR – TerraSAR-X Complex SAR Data Product	142
4.19.1	Driver capabilities	142
4.19.2	See Also	143
4.20	CPG – Convair PolGASP data	143
4.20.1	Driver capabilities	143
4.21	CTable2 – CTable2 Datum Grid Shift	143

4.21.1	Driver capabilities	144
4.22	CTG – USGS LULC Composite Theme Grid	144
4.22.1	Driver capabilities	145
4.23	DAAS (Airbus DS Intelligence Data As A Service driver)	145
4.23.1	Driver capabilities	145
4.23.2	Dataset name syntax	146
4.23.3	Authentication	146
4.23.4	Open options	146
4.24	DB2 raster	147
4.24.1	Driver capabilities	148
4.24.2	Opening options	148
4.24.3	Creation issues	149
4.24.3.1	Tile formats	149
4.24.3.2	Tiling schemes	150
4.24.3.3	Creation options	151
4.24.4	Overviews	151
4.24.5	Metadata	152
4.24.6	Examples	152
4.24.7	See Also	153
4.25	DDS – DirectDraw Surface	153
4.25.1	Driver capabilities	153
4.25.2	Build instructions	154
4.25.2.1	Build crunch	154
4.25.2.2	Build GDAL against crunch	154
4.26	DERIVED – Derived subdatasets driver	154
4.26.1	Available functions	155
4.26.2	Accessing derived subdatasets	155
4.26.3	Listing available subdatasets	157
4.26.4	See Also:	159
4.27	DIMAP – Spot DIMAP	159
4.27.1	Driver capabilities	159
4.28	DIPEX – ELAS DIPEX	159
4.28.1	Driver capabilities	160
4.29	DODS – OPeNDAP Grid Client	160
4.29.1	Driver capabilities	160
4.29.2	Dataset Naming	160
4.29.3	Specialized AIS/DAS Metadata	161
4.29.3.1	Dataset	162
4.29.3.2	Band	162
4.29.4	See Also	163
4.30	DOQ1 – First Generation USGS DOQ	163
4.30.1	Driver capabilities	163
4.31	DOQ2 – New Labelled USGS DOQ	164
4.31.1	Driver capabilities	164
4.32	DTED – Military Elevation Data	164
4.32.1	Driver capabilities	165
4.32.2	Read Issues	165
4.32.2.1	Read speed	165
4.32.2.2	Georeferencing Issues	165
4.32.2.3	Configuration options	166
4.32.2.4	Checksum Issues	166
4.32.3	Creation Issues	166
4.32.4	GeoTransform	166
4.32.5	See Also	166

4.33	E00GRID – Arc/Info Export E00 GRID	166
4.33.1	Driver capabilities	167
4.34	ECRGTOC – ECRG Table Of Contents (TOC.xml)	167
4.34.1	Driver capabilities	167
4.34.2	See Also	168
4.35	ECW – Enhanced Compressed Wavelets (.ecw)	168
4.35.1	Driver capabilities	168
4.35.2	Licensing	169
4.35.3	History	169
4.35.4	Creation Options	169
4.35.5	Creation Options:	169
4.35.6	Configuration Options	170
4.35.6.1	ECW Version 3 Files	170
4.35.6.2	ECWP	171
4.35.6.3	Metadata / Georeferencing	171
4.35.6.4	File Metadata Keys:	171
4.35.7	See Also	172
4.36	EEDAI - Google Earth Engine Data API Image	172
4.36.1	Driver capabilities	172
4.36.2	Dataset name syntax	173
4.36.3	Open options	173
4.36.4	Authentication methods	173
4.36.5	Configuration options	173
4.36.6	Overviews	174
4.36.7	Subdatasets	174
4.36.8	Metadata	174
4.36.9	Pixel encoding	174
4.36.9.1	Examples	174
4.36.10	See Also	174
4.37	EHdr – ESRI .hdr Labelled	175
4.37.1	Driver capabilities	175
4.37.2	See Also	176
4.38	EIR – Erdas Imagine Raw	176
4.38.1	Driver capabilities	176
4.39	ELAS - Earth Resources Laboratory Applications Software	177
4.39.1	Driver capabilities	177
4.39.2	See Also	177
4.40	ENVI – ENVI .hdr Labelled Raster	178
4.40.1	Driver capabilities	178
4.41	Epsilon - Wavelet compressed images	179
4.41.1	Driver capabilities	179
4.41.2	Creation options	179
4.41.3	See Also	180
4.42	ESAT – Envisat Image Product	180
4.42.1	Driver capabilities	180
4.43	ERS – ERMapper .ERS	180
4.43.1	Driver capabilities	181
4.43.2	Creation Issues	181
4.43.3	See Also	182
4.44	EXR – Extended Dynamic Range Image File Format	182
4.44.1	Creation Options	182
4.44.2	Driver capabilities	183
4.45	FAST – EOSAT FAST Format	183
4.45.1	Driver capabilities	184

4.45.2	Data	184
4.45.2.1	FAST-L7A	184
4.45.2.2	IRS-1C/1D	185
4.45.3	Georeference	185
4.45.4	Metadata	185
4.45.5	See Also	186
4.46	FIT – FIT	186
4.46.1	Driver capabilities	186
4.47	FITS – Flexible Image Transport System	186
4.47.1	Notes on CFITSIO linking in GDAL	187
4.47.1.1	Linux	187
4.47.1.2	MacOSX	187
4.47.2	Driver capabilities	188
4.48	FujiBAS – Fuji BAS Scanner Image	188
4.48.1	Driver capabilities	188
4.49	GenBin – Generic Binary (.hdr labelled)	189
4.49.1	Driver capabilities	189
4.50	Oracle Spatial GeoRaster	189
4.50.1	Driver capabilities	190
4.50.2	Browsing the database for GeoRasters	191
4.50.3	Creation Options	191
4.50.4	Importing GeoRaster	193
4.50.5	Exporting GeoRaster	193
4.50.6	Cross schema access	194
4.50.7	General use of GeoRaster	194
4.51	GFF – Sandia National Laboratories GSAT File Format	194
4.51.1	Driver capabilities	195
4.52	GIF – Graphics Interchange Format	195
4.52.1	Driver capabilities	195
4.52.2	Creation Issues	196
4.52.3	See Also	196
4.53	GMT – GMT Compatible netCDF	196
4.53.1	Driver capabilities	197
4.54	GPKG – GeoPackage raster	197
4.54.1	Driver capabilities	198
4.54.2	Opening options	198
4.54.3	Creation issues	199
4.54.3.1	Tile formats	199
4.54.3.2	Tiling schemes	200
4.54.3.3	Nodata value	201
4.54.3.4	Creation options	202
4.54.4	Overviews	203
4.54.5	Metadata	203
4.54.6	Level of support of GeoPackage Extensions	203
4.54.7	Examples	204
4.54.8	See Also	205
4.54.9	Other notes	205
4.55	GRASS Raster Format	205
4.55.1	Driver capabilities	206
4.55.2	Notes on driver variations	206
4.55.3	See Also	206
4.56	GRASSASCIIGrid – GRASS ASCII Grid	207
4.56.1	Driver capabilities	207
4.57	GRIB – WMO General Regularly-distributed Information in Binary form	207

4.57.1	Driver capabilities	208
4.57.2	Configuration options	209
4.57.3	GRIB2 write support	209
4.57.3.1	Product identification and definition	209
4.57.3.2	Data encoding	210
4.57.3.3	Data units	211
4.57.3.4	GRIB2 to GRIB2 conversions	211
4.57.3.5	Examples	212
4.57.4	See Also:	212
4.57.5	Credits	212
4.58	GS7BG – Golden Software Surfer 7 Binary Grid File Format	212
4.58.1	Driver capabilities	212
4.59	GSAG – Golden Software ASCII Grid File Format	213
4.59.1	Driver capabilities	213
4.60	GSBG – Golden Software Binary Grid File Format	214
4.60.1	Driver capabilities	214
4.61	GSC – GSC Geogrid	214
4.61.1	Driver capabilities	215
4.62	GTA - Generic Tagged Arrays	215
4.62.1	Driver capabilities	215
4.62.2	Creation options	216
4.62.3	See Also	216
4.63	GTiff – GeoTIFF File Format	216
4.63.1	Driver capabilities	216
4.63.2	Georeferencing	217
4.63.3	Internal nodata masks	217
4.63.4	Overviews	218
4.63.5	Overviews and nodata masks	218
4.63.6	Metadata	218
4.63.7	Color Profile Metadata	219
4.63.8	Nodata value	220
4.63.9	Sparse files	220
4.63.10	Raw mode	220
4.63.11	Open options	220
4.63.12	Creation Issues	221
4.63.12.1	Creation Options	221
4.63.12.2	Subdatasets	224
4.63.12.3	About JPEG compression of RGB images	224
4.63.12.4	Streaming operations	224
4.63.12.5	Configuration options	225
4.63.13	See Also	226
4.63.13.1	WLD – ESRI World File	226
4.64	GXF – Grid eXchange File	227
4.64.1	Driver capabilities	227
4.65	HDF4 – Hierarchical Data Format Release 4 (HDF4)	227
4.65.1	Driver capabilities	228
4.65.2	Multiple Image Handling (Subdatasets)	228
4.65.3	Georeference	230
4.65.4	Creation Issues	231
4.65.5	Metadata	231
4.65.6	Multidimensional API support	231
4.65.7	Driver building	231
4.65.8	See Also	231
4.66	HDF5 – Hierarchical Data Format Release 5 (HDF5)	232

4.66.1	Driver capabilities	232
4.66.2	Multiple Image Handling (Subdatasets)	232
4.66.3	Georeference	236
4.66.4	Multi-file support	237
4.66.5	Multidimensional API support	237
4.66.6	Driver building	237
4.66.7	See Also	237
4.67	HF2 – HF2/HFZ heightfield raster	237
4.67.1	Driver capabilities	238
4.67.2	Creation options	238
4.67.3	See also	238
4.68	HFA – Erdas Imagine .img	238
4.68.1	Driver capabilities	239
4.68.2	Creation Issues	239
4.68.3	Configuration Options	240
4.68.4	See Also	240
4.69	IDA – Image Display and Analysis	241
4.69.1	Driver capabilities	241
4.70	RST – Idrisi Raster Format	242
4.70.1	Driver capabilities	242
4.70.2	See Also	243
4.71	IGNFHeightASCIIGrid – IGN-France height correction ASCII grids	243
4.71.1	Driver capabilities	243
4.72	ILWIS – Raster Map	244
4.72.1	Driver capabilities	244
4.73	INGR – Intergraph Raster Format	245
4.73.1	Driver capabilities	245
4.73.2	Reading INGR Files	246
4.73.3	Writing INGR Files	246
4.73.4	File Extension	247
4.73.5	Georeference	247
4.73.6	Metadata	247
4.73.7	See Also	247
4.74	IRIS – Vaisala’s weather radar software format	248
4.74.1	Driver capabilities	248
4.75	ISCE – ISCE	249
4.75.1	Driver capabilities	249
4.76	ISG – International Service for the Geoid	250
4.76.1	Driver capabilities	250
4.77	ISIS2 – USGS Astrogeology ISIS Cube (Version 2)	250
4.77.1	Driver capabilities	251
4.77.2	Creation Issues	251
4.77.2.1	Creation Options	251
4.77.3	See Also	252
4.78	ISIS3 – USGS Astrogeology ISIS Cube (Version 3)	252
4.78.1	Driver capabilities	252
4.78.2	Metadata	253
4.78.3	Creation support	255
4.78.4	Examples	256
4.78.5	See Also	257
4.79	JDEM – Japanese DEM (.mem)	257
4.79.1	Driver capabilities	257
4.80	JP2ECW – ERDAS JPEG2000 (.jp2)	258
4.80.1	Driver capabilities	258

4.80.2	Licensing	258
4.80.3	History	259
4.80.4	Georeferencing	259
4.80.5	Option Options	259
4.80.6	Creation Options:	259
4.80.7	Configuration Options	261
4.80.8	Metadata	261
4.80.9	See Also	262
4.81	JP2KAK – JPEG-2000 (based on Kakadu)	262
4.81.1	Driver capabilities	263
4.81.2	Configuration Options	263
4.81.3	Georeferencing	263
4.81.4	Option Options	264
4.81.5	Creation Issues	264
4.81.6	Known Kakadu Issues	265
4.81.6.1	Alpha Channel Writing in v7.8	265
4.81.6.2	kdu_get_num_processors always returns 0 for some platforms	265
4.81.7	See Also	265
4.82	JP2Lura – JPEG2000 driver based on Lurawave library	266
4.82.1	Driver capabilities	266
4.82.2	Georeferencing	266
4.82.3	License number	267
4.82.4	Option Options	267
4.82.5	Creation Options	267
4.82.5.1	Lossless compression	269
4.82.6	Vector information	269
4.82.7	Bugs	270
4.82.8	See Also	270
4.83	JP2MrSID – JPEG2000 via MrSID SDK	270
4.83.1	Driver capabilities	270
4.83.2	Georeferencing	271
4.83.3	Creation Options	271
4.83.4	See Also	272
4.84	JP2OpenJPEG – JPEG2000 driver based on OpenJPEG library	272
4.84.1	Driver capabilities	272
4.84.2	Georeferencing	273
4.84.3	Thread support	273
4.84.4	Option Options	273
4.84.5	Creation Options	274
4.84.5.1	Lossless compression	276
4.84.5.2	GMLJP2v2 definition file	276
4.84.6	Vector information	280
4.84.7	See Also	280
4.85	JPEG2000 – Implementation of the JPEG-2000 part 1	281
4.85.1	Driver capabilities	281
4.85.2	Georeferencing	281
4.85.3	Option Options	282
4.85.4	Creation Options	282
4.85.5	See Also	284
4.86	JPEGLS	284
4.86.1	Driver capabilities	284
4.86.2	Creation Options	285
4.86.3	See Also:	285
4.87	JPEG – JPEG JFIF File Format	285

4.87.1	Driver capabilities	286
4.87.2	Color Profile Metadata	286
4.87.3	Error management	286
4.87.4	Creation Options	286
4.87.5	EXIF and GPS tags	287
4.87.6	See Also	291
4.88	JPIPKAK - JPIP Streaming	291
4.88.1	Driver capabilities	291
4.88.2	JPIPKAK - JPIP Overview	291
4.88.3	JPIPKAK -approach	292
4.88.4	JPIPKAK - implementation	293
4.88.5	JPIPKAK - installation requirements	297
4.88.6	See Also	297
4.88.7	NOTES	297
4.89	KEA	297
4.89.1	Driver capabilities	298
4.89.2	Creation options	298
4.89.3	See Also	299
4.90	KMLSuperoverlay – KMLSuperoverlay	299
4.90.1	Driver capabilities	299
4.91	KRO – KOLOR Raw format	300
4.91.1	Driver capabilities	300
4.92	LAN – Erdas 7.x .LAN and .GIS	300
4.92.1	Driver capabilities	301
4.93	L1B – NOAA Polar Orbiter Level 1b Data Set (AVHRR)	301
4.93.1	Driver capabilities	301
4.93.2	Georeference	302
4.93.3	Data	302
4.93.4	Metadata	302
4.93.5	Subdatasets	303
4.93.6	Nodata mask	303
4.93.7	See Also	303
4.94	LCP – FARSITE v.4 LCP Format	303
4.94.1	Driver capabilities	304
4.94.2	Metadata	304
4.94.2.1	Dataset	304
4.94.2.2	Band	305
4.94.3	Creation Options	305
4.95	Leveller – Daylon Leveller Heightfield	307
4.95.1	Driver capabilities	308
4.95.2	Reading	308
4.95.3	Writing	308
4.95.4	See Also:	309
4.96	LOSLAS – NADCON .los/.las Datum Grid Shift	309
4.96.1	Driver capabilities	309
4.97	MAP – OziExplorer .MAP	309
4.97.1	Driver capabilities	310
4.97.2	See Also:	310
4.98	MRF – Meta Raster Format	310
4.98.1	Driver capabilities	310
4.98.2	Links	311
4.99	MBTiles	311
4.99.1	Driver capabilities	312
4.99.2	Opening options	312

4.99.3	Raster creation issues	313
4.99.3.1	Tile formats	313
4.99.4	Vector creation issues	314
4.99.5	Creation options	314
4.99.6	Layer configuration (vector)	315
4.99.7	Layer creation options (vector)	316
4.99.8	Overviews (raster)	316
4.99.9	Vector tiles	316
4.99.10	Examples:	316
4.99.11	See Also	318
4.100	MEM – In Memory Raster	318
4.100.1	Dataset Name Format	319
4.100.2	Creation Options	319
4.100.3	Driver capabilities	320
4.100.4	Multidimensional API support	320
4.101	MFF – Vexcel MFF Raster	320
4.101.1	Driver capabilities	321
4.102	MFF2 – Vexcel MFF2 Image	321
4.102.1	Driver capabilities	322
4.102.2	Format Details	322
4.102.2.1	MFF2 Top-level Structure	322
4.102.2.2	The “attrib” File	322
4.102.2.3	The “image_data” File	323
4.102.2.4	The “georef” File	323
4.102.2.5	Supported projections	324
4.102.2.6	Recognized ellipsoids	324
4.102.2.7	Explanation of fields	325
4.103	MG4Lidar – MrSID/MG4 LiDAR Compression / Point Cloud View files	325
4.103.1	Example View files (from View Document specification)	326
4.103.1.1	Simplest possible .view file	326
4.103.1.2	Crop the data	326
4.103.1.3	Expose as a bare earth (Max) DEM	327
4.103.1.4	Intensity image	327
4.103.1.5	RGB image	327
4.103.2	Writing not supported	328
4.103.3	Limitations of current implementation	328
4.103.4	See Also:	328
4.103.4.1	Specification for MrSID/MG4 LiDAR View Documents	328
4.104	MrSID – Multi-resolution Seamless Image Database	336
4.104.1	Driver capabilities	336
4.104.2	Metadata	337
4.104.3	Georeference	337
4.104.4	See Also:	337
4.105	MSG – Meteosat Second Generation	337
4.105.1	Driver capabilities	338
4.105.2	Build Instructions	338
4.105.3	Specification of Source Dataset	338
4.105.4	Georeference and Projection	340
4.105.5	See Also	340
4.106	MSGN – Meteosat Second Generation (MSG) Native Archive Format (.nat)	340
4.106.1	Driver capabilities	341
4.107	NDF – NLAPS Data Format	341
4.107.1	Driver capabilities	341
4.108	NetCDF: Network Common Data Form	342

4.108.1	Driver capabilities	342
4.108.2	Multiple Image Handling (Subdatasets)	342
4.108.3	Dimension	345
4.108.4	Georeference	345
4.108.5	Open options	345
4.108.6	Creation Issues	345
4.108.7	GDAL NetCDF Metadata	345
4.108.8	Driver Improvements (GDAL >= 1.9.0)	347
4.108.8.1	Important Changes	347
4.108.8.2	Creation Options	347
4.108.8.3	Configuration Options	348
4.108.9	VSI Virtual File System API support	348
4.108.10	NetCDF-4 groups support on reading (GDAL >= 3.0)	348
4.108.11	Multidimensional API support	348
4.108.12	Driver building	349
4.108.13	See Also:	349
4.109	NGSGEOID - NOAA NGS Geoid Height Grids	349
4.109.1	Driver capabilities	349
4.109.2	See also	350
4.110	NGW – NextGIS Web	350
4.110.1	Driver capabilities	350
4.110.2	Driver	351
4.110.3	Dataset name syntax	351
4.110.4	Configuration options	351
4.110.5	Authentication	352
4.110.6	Open options	352
4.110.7	Create copy options	352
4.110.8	Metadata	353
4.110.9	Examples	353
4.110.10	See also	353
4.111	NITF – National Imagery Transmission Format	353
4.111.1	NITF – Advanced Driver Information	354
4.111.1.1	CGM Segments	354
4.111.1.2	Multi-Image NITF Files	354
4.111.1.3	Text Segments	355
4.111.1.4	TREs	355
4.111.1.5	TREs as xml:TRE	356
4.111.1.6	Raw File / Image Headers	357
4.111.2	Driver capabilities	357
4.111.3	Creation Issues	358
4.111.4	Links	359
4.111.5	Credit	360
4.112	NTv1 – NTv1 Datum Grid Shift	360
4.112.1	Driver capabilities	360
4.113	NTv2 – NTv2 Datum Grid Shift	360
4.113.1	Driver capabilities	361
4.114	NWT_GRD/NWT_GRC – Northwood/Vertical Mapper File Format	361
4.114.1	Driver capabilities (NWT_GRD)	362
4.114.2	Driver capabilities (NWT_GRC)	362
4.114.3	Color Information	362
4.114.4	Metadata	363
4.114.5	Nodata values	363
4.114.5.1	Creation Options	363
4.115	OZI – OZF2/OZFX3 raster	363

4.115.1	Driver capabilities	364
4.115.2	See also	364
4.116	JAXA PALSAR Processed Products	364
4.116.1	Driver capabilities	365
4.116.2	See Also	365
4.117	PAux – PCI .aux Labelled Raw Format	365
4.117.1	Driver capabilities	366
4.118	PCIDSK – PCI Geomatics Database File	366
4.118.1	Creation Options	366
4.118.2	Driver capabilities	367
4.118.3	See Also:	367
4.119	PCRaster – PCRaster raster file format	367
4.119.1	Driver capabilities	368
4.120	PDF – Geospatial PDF	369
4.120.1	Driver capabilities	369
4.120.2	Vector support	369
4.120.3	Metadata	370
4.120.4	Configuration options	370
4.120.4.1	Open Options	370
4.120.5	LAYERS Metadata domain	371
4.120.6	Restrictions	371
4.120.7	Creation Issues (GDAL >= 1.10.0)	371
4.120.7.1	Creation Options	372
4.120.8	Update of existing files	374
4.120.9	Creation of PDF file from a XML composition file (GDAL >= 3.0)	374
4.120.10	Build dependencies	376
4.120.10.1	Unix build	376
4.120.10.2	Poppler	377
4.120.10.3	PoDoFo	377
4.120.10.4	PDFium	377
4.120.11	Examples	377
4.120.12	See also	378
4.121	PDS – Planetary Data System v3	378
4.121.1	Driver capabilities	379
4.121.2	Georeferencing	379
4.121.3	See Also	380
4.122	PDS4 – NASA Planetary Data System (Version 4)	380
4.122.1	Driver capabilities	381
4.122.2	Metadata	381
4.122.3	Open options (vector only)	381
4.122.4	Creation support	382
4.122.5	Layer creation options (vector/table datasets)	383
4.122.6	Subdataset / multiple image support	383
4.122.7	PDS4 raster examples	384
4.122.8	PDS4 vector examples	386
4.122.9	Limitations	387
4.122.10	See Also:	387
4.123	PLMosaic (Planet Labs Mosaics API)	387
4.123.1	Driver capabilities	387
4.123.2	Dataset name syntax	388
4.123.3	Open options	388
4.123.4	Configuration options	388
4.123.5	Location information	388
4.123.5.1	Examples	389

4.123.6 See Also	391
4.124 PNG – Portable Network Graphics	391
4.124.1 Driver capabilities	391
4.124.2 Color Profile Metadata	392
4.125 PNM – Netpbm (.pgm, .ppm)	392
4.125.1 Driver capabilities	393
4.126 PostGISRaster – PostGIS Raster driver	393
4.126.1 Driver capabilities	393
4.126.2 Connecting to a database	394
4.126.2.1 Additional notes	394
4.126.2.2 Performance hints	395
4.126.3 Examples	395
4.126.4 Credits	395
4.126.5 See Also	395
4.127 PHOTOMOD Raster File	395
4.127.1 Driver capabilities	396
4.127.2 See Also	396
4.128 Rasdaman GDAL driver	396
4.128.1 See Also	397
4.129 Rasterlite - Rasters in SQLite DB	397
4.129.1 Driver capabilities	398
4.129.2 Connection string syntax in read mode	398
4.129.3 Creation issues	398
4.129.3.1 Creation options	399
4.129.4 Overviews	399
4.129.5 Performance hints	399
4.129.6 Examples	399
4.129.7 See Also	401
4.130 RasterLite2 - Rasters in SQLite DB	401
4.130.1 Driver capabilities	402
4.130.2 Opening syntax	402
4.130.3 Creation	402
4.130.4 Creation options	403
4.130.5 Examples	403
4.130.6 See Also	404
4.131 R – R Object Data Store	404
4.131.1 Driver capabilities	404
4.131.2 Creation Options	404
4.132 RDA (DigitalGlobe Raster Data Access)	405
4.132.1 Driver capabilities	405
4.132.2 Dataset name syntax	405
4.132.3 Connection String options (optional)	406
4.132.4 Authentication	406
4.132.5 Configuration options	406
4.132.6 ~/.gbdx-config file	406
4.132.7 Caching	407
4.132.8 Open Options	407
4.132.8.1 Examples	407
4.133 RDB - <i>RIEGL</i> Database	408
4.133.1 Driver capabilities	409
4.133.2 Provided Bands	409
4.134 RIK – Swedish Grid Maps	409
4.134.1 Driver capabilities	410
4.135 RMF – Raster Matrix Format	410

4.135.1	Driver capabilities	410
4.135.2	Metadata	411
4.135.3	Open Options	411
4.135.4	Creation Options	411
4.135.5	See Also:	412
4.136	ROI_PAC – ROI_PAC	412
4.136.1	Driver capabilities	412
4.137	RPFTOC – Raster Product Format/RPF (a.toc)	413
4.137.1	Driver capabilities	414
4.138	RRASTER – R Raster	414
4.138.1	Driver capabilities	414
4.138.2	See Also	415
4.139	RS2 – RadarSat 2 XML Product	415
4.139.1	Driver capabilities	416
4.139.2	Data Calibration	416
4.139.3	See Also	416
4.140	SAFE – Sentinel-1 SAFE XML Product	416
4.140.1	Driver capabilities	417
4.140.2	Multiple measurements	417
4.140.3	Examples	417
4.140.4	Data Calibration	421
4.140.5	See Also	421
4.141	SAR_CEOS – CEOS SAR Image	421
4.141.1	Driver capabilities	422
4.142	SAGA – SAGA GIS Binary Grid File Format	422
4.142.1	Driver capabilities	423
4.143	SDTS – USGS SDTS DEM	423
4.143.1	Driver capabilities	424
4.144	ESRI ArcSDE Raster	424
4.144.1	Driver capabilities	424
4.144.2	GDAL ArcSDE Raster driver features	425
4.144.3	Performance considerations	425
4.144.4	Dataset specification	425
4.145	SENTINEL2 – Sentinel-2 Products	426
4.145.1	Driver capabilities	427
4.145.2	Level-1B	427
4.145.3	Level-1C	427
4.145.4	Level-2A	428
4.145.5	Metadata	428
4.145.6	Performance issues for L1C and L2A	428
4.145.7	Open options	429
4.145.8	Examples	429
4.145.9	See Also	434
4.145.10	Credits	434
4.146	SGI – SGI Image Format	434
4.146.1	Driver capabilities	434
4.147	SIGDEM – Scaled Integer Gridded DEM	435
4.147.1	Driver capabilities	435
4.148	SNODAS – Snow Data Assimilation System	436
4.148.1	Driver capabilities	436
4.149	SRP – Standard Product Format (ASRP/USRP) (.gen)	436
4.149.1	Driver capabilities	437
4.150	SRTMHGT – SRTM HGT Format	437
4.150.1	Driver capabilities	438

4.151	Terragen – Terragen™ Terrain File	438
4.151.1	Driver capabilities	438
4.151.2	Reading	439
4.151.3	Writing	439
4.151.4	Roundtripping	439
4.151.5	See Also	439
4.152	TIL – EarthWatch/DigitalGlobe .TIL	440
4.152.1	Driver capabilities	440
4.153	TileDB - TileDB	440
4.153.1	Driver capabilities	441
4.153.2	Creation options	441
4.153.3	See Also	441
4.154	TSX – TerraSAR-X Product	441
4.154.1	Driver capabilities	442
4.155	USGSDEM – USGS ASCII DEM (and CDED)	442
4.155.1	Driver capabilities	442
4.155.2	Creation Issues	443
4.156	VICAR – VICAR	444
4.156.1	Driver capabilities	444
4.156.2	Metadata	445
4.156.3	Binary prefixes	446
4.156.4	Creation support	447
4.156.5	See Also	448
4.157	VRT – GDAL Virtual Format	448
4.157.1	Introduction	448
4.157.2	.vrt Format	449
4.157.2.1	VRTDataset	449
4.157.2.2	VRTRasterBand	450
4.157.3	Overviews	455
4.157.4	.vrt Descriptions for Raw Files	456
4.157.5	Creation of VRT Datasets	457
4.157.6	Using Derived Bands (with pixel functions in C/C++)	459
4.157.6.1	Default Pixel Functions	461
4.157.6.2	Writing Pixel Functions	461
4.157.7	Using Derived Bands (with pixel functions in Python)	463
4.157.7.1	Examples	464
4.157.7.2	Python module path	466
4.157.7.3	Just-in-time compilation	467
4.157.8	Warped VRT	469
4.157.9	Pansharpened VRT	470
4.157.10	Multidimensional VRT	472
4.157.10.1	Multidimensional VRT	473
4.157.11	vrt:// connection string	475
4.157.12	Multi-threading issues	475
4.157.13	Performance considerations	476
4.157.14	Driver capabilities	476
4.158	WCS – OGC Web Coverage Service	476
4.158.1	Driver capabilities	477
4.158.2	Service description file	477
4.158.2.1	Range and dimension subsetting	479
4.158.2.2	Other WCS parameters	479
4.158.2.3	Open options	480
4.158.2.4	The cache	480
4.158.2.5	The WCS: dataset name syntax	480

4.158.2.6 Time	481
4.158.2.7 Examples	481
4.158.3 See Also	481
4.159 WEBP - WEBP	481
4.159.1 Driver capabilities	482
4.159.2 Creation options	482
4.159.3 See Also	482
4.160 WMS – Web Map Services	482
4.160.1 Driver capabilities	483
4.160.2 XML description file	483
4.160.3 Minidrivers	484
4.160.3.1 WMS	484
4.160.3.2 TileService	486
4.160.3.3 WorldWind	486
4.160.3.4 TMS (GDAL 1.7.0 and later)	486
4.160.3.5 OnEarth Tiled WMS (GDAL 1.9.0 and later)	487
4.160.3.6 VirtualEarth (GDAL 1.9.0 and later)	487
4.160.3.7 ArcGIS REST API (GDAL 2.0 and later)	487
4.160.3.8 Internet Imaging Protocol (IIP) (GDAL 2.1 and later)	488
4.160.4 Examples	488
4.160.5 Open syntax	489
4.160.6 Generation of WMS service description XML file	490
4.160.7 See Also	490
4.161 WMTS – OGC Web Map Tile Service	490
4.161.1 Driver capabilities	491
4.161.2 Open syntax	491
4.161.3 Local service description XML file	492
4.161.4 GetFeatureInfo request	494
4.161.5 Generation of WMTS service description XML file	494
4.161.6 See Also	495
4.162 XPM – X11 Pixmap	495
4.162.1 Driver capabilities	495
4.163 XYZ – ASCII Gridded XYZ	496
4.163.1 Creation options	496
4.163.2 Driver capabilities	496
4.163.3 See also	497
4.164 ZMap – ZMap Plus Grid	497
4.164.1 Driver capabilities	497
5 Vector drivers	499
5.1 Aeronav FAA	502
5.1.1 Driver capabilities	502
5.1.2 See Also	502
5.2 AmigoCloud	502
5.2.1 Driver capabilities	503
5.2.2 Dataset name syntax	503
5.2.3 Configuration options	503
5.2.4 Authentication	504
5.2.5 Geometry	504
5.2.6 Filtering	504
5.2.7 Write support	504
5.2.8 Layer creation options	504
5.2.9 Examples	505
5.2.10 See Also	506

5.3	ESRI ArcObjects	506
5.3.1	Overview	506
5.3.2	Requirements	507
5.3.3	Usage	507
5.3.4	Building Notes	507
5.3.5	Known Issues	507
5.4	ARCGEN - Arc/Info Generate	507
5.4.1	Driver capabilities	508
5.4.2	See Also	508
5.5	Arc/Info Binary Coverage	508
5.5.1	Driver capabilities	509
5.5.2	See Also	509
5.6	Arc/Info E00 (ASCII) Coverage	509
5.6.1	Driver capabilities	510
5.6.2	See Also	510
5.7	BNA - Atlas BNA	510
5.7.1	Driver capabilities	511
5.7.2	Creation Issues	511
5.7.3	VSI Virtual File System API support	512
5.7.4	Example	512
5.7.5	See Also	512
5.8	CAD – AutoCAD DWG	512
5.8.1	Driver capabilities	513
5.8.2	Supported Elements	513
5.8.3	See Also	513
5.9	Carto	514
5.9.1	Driver capabilities	514
5.9.2	Dataset name syntax	514
5.9.3	Configuration options	515
5.9.4	Authentication	515
5.9.5	Geometry	515
5.9.6	Filtering	515
5.9.7	Paging	515
5.9.8	Write support	515
5.9.9	SQL	516
5.9.10	Open options	516
5.9.11	Layer creation options	516
5.9.12	Examples	517
5.9.13	See Also	517
5.10	Cloudant – Cloudant	517
5.10.1	Driver capabilities	518
5.10.2	Cloudant vs OGR concepts	518
5.10.3	Dataset name syntax	518
5.10.4	Authentication	518
5.10.5	Filtering	518
5.10.6	Paging	519
5.10.7	Write support	519
5.10.8	Write support and OGR transactions	519
5.10.9	Layer creation options	519
5.10.10	Examples	520
5.10.11	See Also	520
5.11	CouchDB - CouchDB/GeoCouch	520
5.11.1	Driver capabilities	520
5.11.2	CouchDB vs OGR concepts	521

5.11.3	Dataset name syntax	521
5.11.4	Authentication	521
5.11.5	Filtering	521
5.11.6	Paging	521
5.11.7	Write support	522
5.11.8	Write support and OGR transactions	522
5.11.9	Layer creation options	522
5.11.10	Examples	522
5.11.11	See Also	523
5.12	Comma Separated Value (.csv)	523
5.12.1	Driver capabilities	524
5.12.2	Format	524
5.12.3	Reading CSV containing spatial information	525
5.12.3.1	Building point geometries	525
5.12.3.2	Building line geometries	526
5.12.4	Open options	526
5.12.5	Creation Issues	527
5.12.6	VSI Virtual File System API support	528
5.12.6.1	Examples	528
5.12.7	Particular datasources	529
5.12.8	Other Notes	529
5.13	CSW - OGC CSW (Catalog Service for the Web)	529
5.13.1	Driver capabilities	530
5.13.2	Dataset name syntax	530
5.13.3	Filtering	530
5.13.4	Issues	530
5.13.5	Examples	531
5.13.6	See Also	531
5.14	DB2 Spatial	531
5.14.1	Driver capabilities	532
5.14.2	Connecting to a database	532
5.14.3	Layers	532
5.14.4	SQL statements	532
5.14.5	Creation Issues	533
5.14.5.1	Layer Creation Options	533
5.14.5.2	Spatial Index Creation	533
5.14.6	Examples	533
5.15	Microstation DGN	534
5.15.1	Driver capabilities	534
5.15.2	Supported Elements	535
5.15.3	Styling Information	535
5.15.4	Creation Issues	535
5.16	Microstation DGN v8	536
5.16.1	Driver capabilities	537
5.16.2	Supported Elements	537
5.16.3	Styling Information	538
5.16.4	Metadata	538
5.16.5	Creation Issues	538
5.17	DODS/OPeNDAP	539
5.17.1	Driver capabilities	540
5.17.2	Caveats	540
5.17.3	See Also	540
5.18	AutoCAD DWG	540
5.18.1	DWG_INLINE_BLOCKS	541

5.18.2	Building	542
5.19	AutoCAD DXF	542
5.19.1	Driver capabilities	542
5.19.2	Supported Entities	543
5.19.3	DXF_INLINE_BLOCKS	544
5.19.4	3D Extensibility	545
5.19.5	Character Encodings	545
5.19.6	Creation Issues	546
5.19.6.1	Block References	547
5.19.6.2	Layer Definitions	547
5.20	Google Earth Engine Data API	548
5.20.1	Driver capabilities	548
5.20.2	Dataset name syntax	548
5.20.3	Open options	548
5.20.4	Authentication methods	548
5.20.5	Configuration options	549
5.20.6	Attributes	549
5.20.6.1	Geometry	551
5.20.6.2	Filtering	551
5.20.6.3	Paging	551
5.20.6.4	Extent and feature count	551
5.20.6.5	Examples	551
5.20.7	See Also:	552
5.21	EDIGEO	552
5.21.1	Driver capabilities	552
5.21.2	Labels	553
5.21.3	See Also	553
5.22	Elasticsearch: Geographically Encoded Objects for Elasticsearch	553
5.22.1	Driver capabilities	554
5.22.2	Opening dataset name syntax	554
5.22.3	Elasticsearch vs OGR concepts	555
5.22.4	Field definitions	555
5.22.5	Geometry types	555
5.22.6	Filtering	555
5.22.7	Paging	556
5.22.8	Schema	556
5.22.9	Feature ID	556
5.22.10	ExecuteSQL() interface	556
5.22.11	Getting metadata	557
5.22.12	Write support	557
5.22.13	Spatial reference system	557
5.22.14	Layer creation options	557
5.22.15	Examples	559
5.22.16	See Also	560
5.23	ESRIJSON / FeatureService driver	560
5.23.1	Driver capabilities	561
5.23.2	Datasource	561
5.23.3	Open options	561
5.23.4	Example	561
5.23.5	See Also	562
5.24	ESRI File Geodatabase (FileGDB)	562
5.24.1	Driver capabilities	562
5.24.2	Requirements	562
5.24.3	Bulk feature loading (OGR >= 1.9.2)	563

5.24.4	SQL support (OGR >= 1.10)	563
5.24.4.1	Special SQL requests	563
5.24.5	Transaction support (OGR >= 2.0)	563
5.24.6	CreateFeature() support	563
5.24.7	Dataset Creation Options	563
5.24.8	Layer Creation Options	564
5.24.9	Examples	565
5.24.10	Building Notes	565
5.24.11	Known Issues	565
5.24.12	Links	565
5.25	FlatGeobuf	565
5.25.1	Driver capabilities	566
5.25.2	Multi layer support	566
5.25.3	Open options	566
5.25.4	Dataset Creation Options	566
5.25.5	Layer Creation Options	566
5.25.6	Examples	567
5.25.7	See Also	567
5.26	FMEObjects Gateway	567
5.26.1	Caching	568
5.26.2	Caveats	568
5.26.3	Build/Configuration	568
5.27	GeoConcept text export	568
5.27.1	Driver capabilities	569
5.27.2	GeoConcept Text File Format (gxt)	569
5.27.3	Creation Issues	569
5.27.3.1	Dataset Creation Options	570
5.27.3.2	Layer Creation Options	571
5.27.3.3	Examples	571
5.27.3.4	See Also	573
5.28	GeoJSON	573
5.28.1	Driver capabilities	574
5.28.2	Datasource	574
5.28.3	Layer	574
5.28.4	Feature	575
5.28.5	Geometry	575
5.28.6	Environment variables	575
5.28.7	Open options	576
5.28.8	Layer creation options	577
5.28.9	VSI Virtual File System API support	577
5.28.10	Round-tripping of extra JSon members	577
5.28.11	RFC 7946 write support	578
5.28.12	Examples	578
5.28.13	See Also	579
5.29	GeoJSONSeq: sequence of GeoJSON features	579
5.29.1	Driver capabilities	579
5.29.2	Datasource	580
5.29.3	Layer creation options	580
5.29.4	See Also	580
5.30	Geomedia MDB database	580
5.30.1	Driver capabilities	581
5.30.2	How to use Geomedia driver with unixODBC and MDB Tools (on Unix and Linux)	581
5.30.3	See also	581
5.31	GeoRSS : Geographically Encoded Objects for RSS feeds	581

5.31.1	Driver capabilities	582
5.31.2	Encoding issues	582
5.31.3	Field definitions	583
5.31.4	Creation Issues	584
5.31.5	VSI Virtual File System API support	585
5.31.6	Example	585
5.31.7	See Also	586
5.32	GMLAS - Geography Markup Language (GML) driven by application schemas	586
5.32.1	Driver capabilities	587
5.32.2	Opening syntax	587
5.32.3	Mapping of XML structure to OGR layers and fields	587
5.32.4	Metadata layers	588
5.32.5	Configuration file	588
5.32.6	Geometry support	589
5.32.7	Performance issues with large multi-layer GML files.	589
5.32.8	Open options	589
5.32.9	Creation support	590
5.32.9.1	ogr2ogr behaviour	590
5.32.9.2	Dataset creation options	591
5.32.10	Examples	592
5.32.11	See Also	592
5.32.12	Credits	592
5.32.12.1	GMLAS - Mapping examples	593
5.32.12.2	GMLAS - Metadata layers	596
5.33	GML - Geography Markup Language	599
5.33.1	Driver capabilities	600
5.33.2	Parsers	600
5.33.3	CRS support	600
5.33.4	Schema	601
5.33.5	Particular GML application schemas	601
5.33.6	Geometry reading	602
5.33.7	gml:xlink resolving	602
5.33.8	TopoSurface interpretation rules [polygons and internal holes]	603
5.33.9	Encoding issues	604
5.33.10	Feature id (fid / gml:id)	604
5.33.11	Performance issues with large multi-layer GML files.	604
5.33.12	Open options	605
5.33.13	Creation Issues	606
5.33.14	VSI Virtual File System API support	607
5.33.15	Syntax of .gfs file by example	608
5.33.16	Advanced .gfs syntax (OGR >= 1.11)	609
5.33.16.1	Specifying ElementPath to find objects embedded into top level objects	609
5.33.16.2	Getting XML attributes as OGR fields	610
5.33.16.3	Using conditions on XML attributes	611
5.33.17	Registry for GML application schemas (OGR >= 1.11)	613
5.33.18	Building junction tables	614
5.33.19	Reading datasets resulting from a WFS 2.0 join queries	615
5.33.20	Examples	616
5.33.21	See Also	616
5.33.22	Credits	616
5.34	GMT ASCII Vectors (.gmt)	617
5.34.1	Driver capabilities	617
5.34.2	Creation Issues	617
5.35	GPKG – GeoPackage vector	618

5.35.1	Driver capabilities	618
5.35.2	Specification version	619
5.35.3	Limitations	619
5.35.4	SQL	619
5.35.4.1	SQL functions	619
5.35.4.2	Link with Spatialite	620
5.35.5	Transaction support (GDAL >= 2.0)	620
5.35.6	Opening options	620
5.35.7	Creation Issues	620
5.35.7.1	Dataset Creation Options	620
5.35.7.2	Layer Creation Options	621
5.35.8	Metadata	621
5.35.8.1	Non-spatial tables	622
5.35.9	Spatial views	622
5.35.10	Level of support of GeoPackage Extensions	623
5.35.11	Examples	623
5.35.12	See Also	623
5.35.12.1	GeoPackage aspatial extension	624
5.36	GPSTables	626
5.36.1	Driver capabilities	626
5.36.2	Read support	626
5.36.3	Write support	627
5.36.3.1	Examples	628
5.36.3.2	See Also	628
5.37	GPX - GPS Exchange Format	628
5.37.1	Driver capabilities	629
5.37.2	Encoding issues	629
5.37.3	Extensions element reading	630
5.37.4	Creation Issues	630
5.37.5	Issues when translating to Shapefile	631
5.37.6	VSI Virtual File System API support	632
5.37.7	Example	632
5.37.8	FAQ	633
5.37.9	See Also	633
5.38	GRASS Vector Format	633
5.38.1	Driver capabilities	634
5.38.2	Datasource name	634
5.38.3	Layer names	634
5.38.4	Attribute filter	634
5.38.5	Spatial filter	635
5.38.6	GISBASE	635
5.38.7	Missing topology	635
5.38.8	Random access	635
5.38.9	Known problem	635
5.38.10	See Also	635
5.39	GTM - GPS TrackMaker	635
5.39.1	Driver capabilities	636
5.39.2	Example	636
5.39.3	See Also	637
5.40	HTF - Hydrographic Transfer Format	637
5.40.1	Driver capabilities	638
5.40.2	See Also	639
5.41	IDB	639
5.41.1	Driver capabilities	639

5.41.2	Environment variables	640
5.41.3	Example	640
5.42	Idrisi Vector (.VCT)	640
5.42.1	Driver capabilities	641
5.43	“INTERLIS 1” and “INTERLIS 2” drivers	641
5.43.1	Driver capabilities	642
5.43.2	Model support	642
5.43.2.1	Arc interpolation	643
5.43.3	Other Notes	643
5.44	INGRES	643
5.44.1	Driver capabilities	644
5.44.2	Caveats	644
5.44.3	Creation Issues	644
5.44.3.1	Layer Creation Options	645
5.44.4	Older Versions	645
5.45	JML: OpenJUMP JML format	645
5.45.1	Driver capabilities	646
5.45.2	Encoding issues	646
5.45.3	Styling	646
5.45.4	Creation Issues	647
5.45.5	See Also	647
5.45.6	Credits	647
5.46	KML - Keyhole Markup Language	647
5.46.1	Driver capabilities	647
5.46.1.1	KML Reading	648
5.46.1.2	KML Writing	648
5.46.1.3	Encoding issues	648
5.46.1.4	Creation Options	649
5.46.2	VSI Virtual File System API support	649
5.46.3	Example	649
5.46.4	Caveats	649
5.46.5	See Also	650
5.47	LIBKML Driver (.kml .kmz)	650
5.47.1	Driver capabilities	650
5.47.2	Datasource	651
5.47.2.1	StyleTable	651
5.47.2.2	Datasource creation options	651
5.47.3	Layer	652
5.47.3.1	Style	652
5.47.3.2	Schema	653
5.47.3.3	Layer creation options	653
5.47.4	Feature	654
5.47.4.1	Style	655
5.47.5	Fields	655
5.47.6	Geometry	658
5.47.7	VSI Virtual File System API support	658
5.47.8	Example	658
5.48	MapML	660
5.48.1	Driver capabilities	661
5.48.2	Read support	661
5.48.3	Write support	661
5.48.4	Dataset creation options	661
5.48.5	Links	662
5.49	Access MDB databases	662

5.49.1	How to build the MDB driver (on Linux)	663
5.49.2	How to run the MDB driver (on Linux)	663
5.49.3	Resources	663
5.49.4	See also	663
5.50	Memory	663
5.50.1	Driver capabilities	664
5.50.2	Creation Issues	664
5.51	MapInfo TAB and MIF/MID	664
5.51.1	Driver capabilities	665
5.51.2	Creation Issues	665
5.51.2.1	Dataset Creation Options	666
5.51.2.2	Layer Creation Options	666
5.51.2.3	Configuration options	667
5.51.2.4	See Also	667
5.52	MongoDB	667
5.52.1	Driver capabilities	667
5.52.2	MongoDB vs OGR concepts	668
5.52.3	Dataset name syntax	668
5.52.4	Filtering	669
5.52.5	Paging	669
5.52.6	Schema	669
5.52.7	Feature ID	670
5.52.8	ExecuteSQL() interface	670
5.52.9	Write support	670
5.52.10	Layer creation options	670
5.52.11	Examples	671
5.52.12	Build instructions	671
5.52.13	See Also	672
5.53	MongoDBv3	672
5.53.1	Driver capabilities	672
5.53.2	MongoDB vs OGR concepts	673
5.53.3	Dataset name syntax	673
5.53.4	Filtering	674
5.53.5	Paging	674
5.53.6	Schema	674
5.53.7	Feature ID	674
5.53.8	ExecuteSQL() interface	674
5.53.9	Write support	675
5.53.10	Layer creation options	675
5.53.11	Examples	675
5.53.12	Build instructions	675
5.53.13	See Also	676
5.54	MSSQLSpatial - Microsoft SQL Server Spatial Database	676
5.54.1	Driver capabilities	676
5.54.2	Connecting to a database	677
5.54.3	Layers	677
5.54.4	SQL statements	677
5.54.5	Creation Issues	678
5.54.5.1	Layer Creation Options	678
5.54.5.2	Spatial Index Creation	679
5.54.6	Configuration options	679
5.54.7	Transaction support (GDAL >= 2.0)	679
5.54.8	Examples	680
5.55	MVT: Mapbox Vector Tiles	680

5.55.1	Driver capabilities	681
5.55.2	Connection strings	681
5.55.3	metadata.json	682
5.55.4	Opening options	683
5.55.5	Creation issues	683
5.55.6	Dataset creation options	683
5.55.7	Layer configuration	684
5.55.8	Layer creation options	685
5.55.9	Examples	685
5.56	MySQL	686
5.56.1	Driver capabilities	686
5.56.2	Caveats	686
5.56.3	Creation Issues	687
5.56.3.1	Layer Creation Options	687
5.57	NAS - ALKIS	688
5.57.1	Driver capabilities	689
5.58	NetCDF: Network Common Data Form - Vector	689
5.58.1	Driver capabilities	690
5.58.2	Conventions and Data Formats	690
5.58.2.1	Distinguishing the Two Formats	690
5.58.2.2	CF-1.8 Writing Limitations	690
5.58.3	Mapping of concepts	691
5.58.3.1	Field types	691
5.58.3.2	Layers	692
5.58.3.3	Strings	692
5.58.3.4	Geometry	693
5.58.3.5	“Profile” feature type	693
5.58.4	Dataset creation options	694
5.58.5	Layer creation options	694
5.58.6	XML configuration file	695
5.58.7	Further Reading	696
5.58.8	Credits	696
5.59	NGW – NextGIS Web	696
5.59.1	Driver capabilities	697
5.59.2	Driver	697
5.59.3	Dataset name syntax	697
5.59.4	Configuration options	698
5.59.5	Authentication	698
5.59.6	Feature	698
5.59.7	Geometry	698
5.59.8	Field data types	699
5.59.9	Paging	699
5.59.10	Write support	699
5.59.11	Open options	699
5.59.12	Dataset creation options	699
5.59.13	Layer creation options	700
5.59.14	Metadata	700
5.59.15	Filters	700
5.59.16	Examples	701
5.59.17	See also	702
5.60	UK .NTF	702
5.60.1	Driver capabilities	702
5.60.2	See Also	703
5.60.3	Implementation Notes	703

5.60.3.1	Products (and Layers) Supported	703
5.60.3.2	Product Schemas	704
5.60.3.3	Special Attributes	704
5.60.3.4	Generic Products	705
5.61	OGC API - Features	706
5.61.1	Driver capabilities	706
5.61.2	Dataset name syntax	706
5.61.3	Layer schema	706
5.61.4	Filtering	706
5.61.5	Open options	707
5.61.6	Examples	707
5.61.7	See Also	709
5.62	Oracle Spatial	709
5.62.1	Driver capabilities	709
5.62.2	SQL Issues	710
5.62.3	Caveats	710
5.62.4	Creation Issues	710
5.62.4.1	Layer Creation Options	711
5.62.4.2	Layer Open Options	712
5.62.4.3	Example	712
5.62.4.4	Credits	712
5.63	ODBC RDBMS	713
5.63.1	Driver capabilities	714
5.63.2	Access Databases (.MDB) support	714
5.63.3	Creation Issues	714
5.63.3.1	See Also	714
5.64	ODS - Open Document Spreadsheet	714
5.64.1	Driver capabilities	715
5.64.2	Configuration options	715
5.65	OGDI Vectors	715
5.65.1	Driver capabilities	716
5.65.2	Error handling	716
5.65.3	Examples	716
5.65.4	See Also	717
5.66	OpenAir - OpenAir Special Use Airspace Format	717
5.66.1	Driver capabilities	717
5.66.2	See Also	718
5.67	ESRI File Geodatabase (OpenFileGDB)	718
5.67.1	Driver capabilities	718
5.67.2	Spatial filtering	718
5.67.3	SQL support	719
5.67.3.1	Special SQL requests	719
5.67.4	Comparison with the FileGDB driver	719
5.67.5	Examples	719
5.67.6	Links	720
5.68	OSM - OpenStreetMap XML and PBF	720
5.68.1	Driver capabilities	720
5.68.2	Configuration	721
5.68.2.1	“other_tags” field	721
5.68.2.2	“all_tags” field	721
5.68.3	Internal working and performance tweaking	721
5.68.4	Interleaved reading	722
5.68.5	Spatial filtering	722
5.68.6	Reading .osm.bz2 files and/or online files	722

5.68.7	Open options	723
5.68.8	See Also	723
5.69	PDF – Geospatial PDF	723
5.69.1	Driver capabilities	724
5.69.2	Vector support	724
5.69.3	Feature style support	724
5.69.4	See Also	725
5.70	PDS - Planetary Data Systems TABLE	725
5.70.1	Driver capabilities	726
5.70.2	See Also	726
5.71	PostgreSQL SQL Dump	726
5.71.1	Driver capabilities	726
5.71.2	Creation options	727
5.71.2.1	Dataset Creation Options	727
5.71.2.2	Layer Creation Options	727
5.71.2.3	Environment variables	728
5.71.2.4	VSI Virtual File System API support	728
5.71.2.5	Example	729
5.71.2.6	See Also	729
5.72	ESRI Personal GeoDatabase	729
5.72.1	Driver capabilities	730
5.72.2	How to use PGeo driver with unixODBC and MDB Tools (on Unix and Linux)	730
5.72.2.1	Prerequisites	730
5.72.2.2	Configuration	730
5.72.2.3	Testing PGeo driver with ogrinfo	732
5.72.3	Resources	732
5.72.4	See also	732
5.73	PostgreSQL / PostGIS	732
5.73.1	Driver capabilities	733
5.73.2	Connecting to a database	733
5.73.3	Geometry columns	733
5.73.4	SQL statements	734
5.73.5	Creation Issues	734
5.73.5.1	Dataset open options	734
5.73.5.2	Dataset Creation Options	735
5.73.5.3	Layer Creation Options	735
5.73.5.4	Configuration Options	736
5.73.5.5	Examples	736
5.73.6	FAQs	738
5.73.7	See Also	738
5.73.7.1	PostgreSQL / PostGIS - Advanced Driver Information	738
5.74	PLScenes (Planet Labs Scenes/Catalog API)	743
5.74.1	PLScenes (Planet Labs Scenes), Data V1 API	744
5.74.1.1	Driver capabilities	744
5.74.1.2	Dataset name syntax	744
5.74.1.3	Open options	744
5.74.1.4	Configuration options	745
5.74.1.5	Attributes	745
5.74.1.6	Raster access	746
5.74.1.7	See Also	747
5.74.2	See Also	747
5.75	IHO S-57 (ENC)	747
5.75.1	Driver capabilities	748
5.75.2	Feature Translation	748

5.75.3	Soundings	749
5.75.4	S57 Control Options	749
5.75.5	S-57 Export	750
5.75.6	See Also	751
5.76	ESRI ArcSDE	751
5.76.1	Layer Creation Options	752
5.76.2	Environment variables	752
5.76.3	Examples	753
5.77	SDTS	753
5.77.1	Driver capabilities	753
5.77.2	See Also	754
5.78	SEG-P1 / UKOOA P1/90	754
5.78.1	Driver capabilities	754
5.78.2	See Also	754
5.79	SEG-Y / SEG-Y	754
5.79.1	Driver capabilities	755
5.79.2	See Also	755
5.80	Selafin files	755
5.80.1	Driver capabilities	756
5.80.2	Magic bytes	756
5.80.3	Format	756
5.80.3.1	Elements	756
5.80.3.2	Full structure	758
5.80.4	Mapping between file and layers	759
5.80.4.1	Layers in a Selafin datasource	759
5.80.4.2	Constraints on layers	759
5.80.5	Layer filtering specification	760
5.80.6	Datasource creation options	760
5.80.7	Layer creation options	760
5.80.8	Notes about the creation and the update of a Selafin datasource	761
5.80.9	VSI Virtual File System API support	762
5.80.10	Other notes	762
5.81	ESRI Shapefile / DBF	762
5.81.1	Encoding	763
5.81.2	Open options	763
5.81.3	Spatial and Attribute Indexing	764
5.81.4	Creation Issues	764
5.81.5	Field sizes	765
5.81.6	Spatial extent	765
5.81.7	Size Issues	766
5.81.8	Dataset Creation Options	766
5.81.9	Layer Creation Options	766
5.81.10	VSI Virtual File System API support	767
5.81.11	Compressed files	767
5.81.12	Examples	767
5.81.13	Advanced topics	767
5.81.14	Driver capabilities	767
5.81.15	See Also	768
5.82	Norwegian SOSI Standard	768
5.82.1	Open options	768
5.82.1.1	Examples	768
5.83	SQLite / Spatialite RDBMS	769
5.83.1	Driver capabilities	770
5.83.2	“Regular” SQLite databases	770

5.83.3	Tables with multiple geometry columns	770
5.83.4	REGEXP operator	770
5.83.5	Using the Spatialite library (Spatial extension for SQLite)	771
5.83.6	Opening with ‘VirtualShape:’	771
5.83.7	The SQLite SQL dialect	771
5.83.8	The VirtualOGR SQLite extension	771
5.83.9	Creation Issues	772
5.83.9.1	Transaction support (GDAL >= 2.0)	772
5.83.9.2	Dataset open options	772
5.83.9.3	Database Creation Options	773
5.83.9.4	Layer Creation Options	773
5.83.10	Other Configuration Options	774
5.83.11	Performance hints	774
5.83.12	Example	775
5.83.13	Credits	775
5.83.14	Links	775
5.84	SUA - Tim Newport-Peace’s Special Use Airspace Format	775
5.84.1	Driver capabilities	776
5.84.2	See Also	776
5.85	SVG - Scalable Vector Graphics	776
5.85.1	Driver capabilities	776
5.85.2	See Also	777
5.86	Storage and eXchange Format - SXF	777
5.86.1	Driver capabilities	778
5.86.2	See Also	778
5.87	U.S. Census TIGER/Line	778
5.87.1	Driver capabilities	779
5.87.2	Feature Representation	779
5.87.2.1	CompleteChain	780
5.87.2.2	AltName	780
5.87.2.3	FeatureIds	780
5.87.2.4	ZipCodes	780
5.87.2.5	Landmarks	781
5.87.2.6	AreaLandmarks	781
5.87.2.7	KeyFeatures	781
5.87.2.8	Polygon	781
5.87.2.9	EntityNames	781
5.87.2.10	IDHistory	781
5.87.2.11	PolyChainLink	782
5.87.2.12	PIP	782
5.87.2.13	ZipPlus4	782
5.87.3	See Also	782
5.88	TopoJSON driver	782
5.88.1	Driver capabilities	783
5.88.2	Datasource	783
5.88.3	See Also	783
5.89	VDV - VDV-451/VDV-452/INTREST Data Format	783
5.89.1	Driver capabilities	784
5.89.2	Creations issues	784
5.89.3	Links	785
5.90	VFK - Czech Cadastral Exchange Data Format	785
5.90.1	Driver capabilities	785
5.90.2	Open options	786
5.90.2.1	Configuration options	786

5.90.2.2	Internal working and performance tweaking	786
5.90.3	Datasource name	786
5.90.4	Layer names	787
5.90.5	Filters	787
5.90.5.1	Attribute filter	787
5.90.5.2	Spatial filter	787
5.90.6	References	787
5.91	VRT – Virtual Format	787
5.91.1	Driver capabilities	788
5.91.2	Creation Issues	788
5.91.3	Virtual File Format	788
5.91.4	Example: ODBC Point Layer	793
5.91.5	Example: Renaming attributes	793
5.91.6	Example: Transparent spatial filtering (GDAL >= 1.7.0)	794
5.91.7	Example: Reprojected layer (GDAL >= 1.10.0)	794
5.91.8	Example: Union layer (GDAL >= 1.10.0)	794
5.91.9	Example: SQLite/Spatialite SQL dialect (GDAL >= 1.10.0)	794
5.91.10	Example: Multiple geometry fields (GDAL >= 1.11)	795
5.91.11	Other Notes	796
5.92	Walk - Walk Spatial Data	796
5.92.1	Driver capabilities	797
5.92.2	How to use Walk driver with unixODBC and MDB Tools (on Unix and Linux)	797
5.93	WAsP - WAsP .map format	797
5.93.1	Driver capabilities	797
5.93.2	Configuration options	798
5.94	WFS - OGC WFS service	798
5.94.1	Driver capabilities	799
5.94.2	Dataset name syntax	799
5.94.3	Request paging	800
5.94.4	Filtering	800
5.94.5	Layer joins	801
5.94.6	Write support / WFS-T	801
5.94.7	Write support and OGR transactions	802
5.94.8	Special SQL commands	802
5.94.9	Special layer : WFSLayerMetadata	802
5.94.10	Special layer : WFSGetCapabilities	802
5.94.11	Open options (GDAL >= 2.0)	803
5.94.12	Examples	803
5.94.13	See Also	804
5.95	XLS - MS Excel format	804
5.95.1	Configuration options	804
5.95.2	See Also	804
5.96	XLSX - MS Office Open XML spreadsheet	804
5.96.1	Driver capabilities	805
5.96.2	Configuration options	805
5.97	X-Plane/Flightgear aeronautical data	805
5.97.1	Driver capabilities	806
5.97.2	Examples	806
5.97.3	See Also	807
5.97.4	Airport data (apt.dat)	807
5.97.4.1	APT layer	807
5.97.4.2	RunwayThreshold layer	808
5.97.4.3	RunwayPolygon layer	810
5.97.4.4	WaterRunwayThreshold (Point)	810

5.97.4.5	WaterRunwayPolygon (Polygon)	810
5.97.4.6	Stopway layer (Polygon)	811
5.97.4.7	Helipad (Point)	811
5.97.4.8	HelipadPolygon (Polygon)	811
5.97.4.9	TaxiwayRectangle (Polygon) - V810 record	812
5.97.4.10	Pavement (Polygon)	812
5.97.4.11	APTBoundary (Polygon)	812
5.97.4.12	APTLinearFeature (Line String)	812
5.97.4.13	StartupLocation (Point)	813
5.97.4.14	APTLightBeacon (Point)	813
5.97.4.15	APTWindsock (Point)	813
5.97.4.16	TaxiwaySign (Point)	813
5.97.4.17	VASI_PAPI_WIGWAG (Point)	814
5.97.4.18	ATCFreq (None)	814
5.97.5	Navigation aids (nav.dat)	815
5.97.5.1	ILS (Point)	815
5.97.5.2	VOR (Point)	816
5.97.5.3	NDB (Point)	816
5.97.5.4	GS - Glideslope (Point)	816
5.97.5.5	Marker - ILS marker beacons. (Point)	817
5.97.5.6	DME (Point)	817
5.97.5.7	DMEILS (Point)	817
5.97.6	IFR intersections (fix.dat)	818
5.97.6.1	FIX (Point)	818
5.97.7	Airways (awy.dat)	818
5.97.7.1	AirwaySegment (Line String)	818
5.97.7.2	AirwayIntersection (Point)	818
6	User oriented documentation	819
6.1	Raster Data Model	819
6.1.1	Dataset	819
6.1.2	Coordinate System	819
6.1.3	Affine GeoTransform	820
6.1.4	GCPs	820
6.1.5	Metadata	821
6.1.5.1	SUBDATASETS Domain	822
6.1.5.2	IMAGE_STRUCTURE Domain	822
6.1.5.3	RPC Domain	823
6.1.5.4	IMAGERY Domain (remote sensing)	823
6.1.5.5	xml: Domains	824
6.1.6	Raster Band	824
6.1.7	Color Table	825
6.1.8	Overviews	826
6.2	Multidimensional Raster Data Model	826
6.2.1	Group	826
6.2.2	Attribute	826
6.2.3	Multidimensional array	826
6.2.4	Dimension	827
6.2.5	Data Type	827
6.2.6	Differences with the GDAL 2D raster data model	827
6.2.7	Bridges between GDAL 2D classic raster data model and multidimensional data model	828
6.2.8	Applications	828
6.3	Vector Data Model	828
6.3.1	Class Overview	828

6.3.2	Geometry	829
6.3.2.1	Compatibility issues with GDAL 2.0 non-linear geometries	829
6.3.3	Spatial Reference	829
6.3.4	Feature / Feature Definition	830
6.3.5	Layer	830
6.3.6	Dataset	831
6.3.7	Drivers	831
6.4	Geographic Networks Data Model	831
6.4.1	General concept	831
6.4.1.1	Network	832
6.4.1.2	Format	832
6.4.2	Network formats	832
6.4.2.1	GNMGenericNetwork	832
6.4.3	Network analysis	833
6.5	OGR SQL dialect and SQLITE SQL dialect	833
6.5.1	OGR SQL dialect	833
6.5.1.1	SELECT	833
6.5.1.2	SPECIAL FIELDS	840
6.5.1.3	CREATE INDEX	842
6.5.1.4	DROP INDEX	842
6.5.1.5	ALTER TABLE	842
6.5.1.6	DROP TABLE	843
6.5.1.7	ExecuteSQL()	843
6.5.1.8	Non-OGR SQL	843
6.5.2	SQL SQLite dialect	843
6.5.2.1	SELECT statement	844
6.6	GDAL Virtual File Systems (compressed, network hosted, etc. . .): /vsimem, /vsizip, /vsitar, /vsicurl,	849
6.6.1	Introduction	849
6.6.2	Chaining	849
6.6.3	Drivers supporting virtual file systems	849
6.6.4	/vsizip/ (.zip archives)	849
6.6.5	/vsigzip/ (gzipped file)	850
6.6.6	/vsitar/ (.tar, .tgz archives)	851
6.6.7	Network based file systems	851
6.6.7.1	/vsicurl/ (http/https/ftp files: random access)	852
6.6.7.2	/vsicurl_streaming/ (http/https/ftp files: streaming)	853
6.6.7.3	/vsis3/ (AWS S3 files: random reading)	853
6.6.7.4	/vsis3_streaming/ (AWS S3 files: streaming)	854
6.6.7.5	/vsigs/ (Google Cloud Storage files: random reading)	855
6.6.7.6	/vsigs_streaming/ (Google Cloud Storage files: streaming)	856
6.6.7.7	/vsiaz/ (Microsoft Azure Blob files: random reading)	856
6.6.7.8	/vsiaz_streaming/ (Microsoft Azure Blob files: streaming)	857
6.6.7.9	/vsioss/ (Alibaba Cloud OSS files: random reading)	857
6.6.7.10	/vsioss_streaming/ (Alibaba Cloud OSS files: streaming)	857
6.6.7.11	/vsiswift/ (OpenStack Swift Object Storage: random reading)	858
6.6.7.12	/vsiswift_streaming/ (OpenStack Swift Object Storage: streaming)	859
6.6.7.13	/vsihdfs/ (Hadoop File System)	859
6.6.7.14	/vsiwebhdfs/ (Web Hadoop File System REST API)	859
6.6.8	/vsistdin/ (standard input streaming)	860
6.6.9	/vsistdout/ (standard output streaming)	860
6.6.10	/vsimem/ (in-memory files)	860
6.6.11	/vsisubfile/ (portions of files)	860
6.6.12	/vsiparse/ (sparse files)	861

6.6.13	File caching	862
6.6.14	/vsicrypt/ (encrypted files)	862
6.7	Feature Style Specification	862
6.7.1	1. Overview	862
6.7.1.1	1.1 Style is a property of Feature object	862
6.7.1.2	1.2 Feature Styles can be stored at 2 levels	863
6.7.1.3	1.3 Drawing Tools	863
6.7.1.4	1.4 Feature attributes can be used by style definitions	863
6.7.1.5	1.5 Tool parameter units	864
6.7.2	2. Feature Style String	864
6.7.2.1	2.1 Examples	864
6.7.2.2	2.2 Style String Syntax	865
6.7.2.3	2.3 Pen Tool Parameters	866
6.7.2.4	2.4 Brush Tool Parameters	868
6.7.2.5	2.5 Symbol Tool Parameters	870
6.7.2.6	2.6 Label Tool Parameters	872
6.7.2.7	2.7 Styles Table Format	874
6.7.2.8	2.8 Using OGR SQL to transfer the style between the data sources	874
6.7.3	3. OGR Support Classes	874
6.7.4	REVISION HISTORY	876
6.8	Configuration options	877
6.8.1	How to set configuration options ?	877
6.8.2	List of configuration options and where they apply	878
7	API	881
8	Tutorials	883
8.1	Raster	883
8.1.1	Raster API tutorial	883
8.1.1.1	Opening the File	883
8.1.1.2	Getting Dataset Information	884
8.1.1.3	Fetching a Raster Band	885
8.1.1.4	Reading Raster Data	887
8.1.1.5	Closing the Dataset	888
8.1.1.6	Techniques for Creating Files	888
8.1.1.7	Using CreateCopy()	889
8.1.1.8	Using Create()	891
8.1.2	Raster driver implementation tutorial	892
8.1.2.1	Overall Approach	892
8.1.2.2	Implementing the Dataset	892
8.1.2.3	Implementing the RasterBand	896
8.1.2.4	The Driver	897
8.1.2.5	Adding Driver to GDAL Tree	899
8.1.2.6	Adding Georeferencing	899
8.1.2.7	Overviews	900
8.1.2.8	File Creation	901
8.1.2.9	RawDataset/RawRasterBand Helper Classes	905
8.1.2.10	Metadata, and Other Exotic Extensions	906
8.1.3	GDAL Warp API tutorial (Reprojection, ...)	907
8.1.3.1	Overview	907
8.1.3.2	A Simple Reprojection Case	907
8.1.3.3	Other Warping Options	908
8.1.3.4	Creating the Output File	909
8.1.3.5	Performance Optimization	910

8.1.3.6	Other Masking Options	911
8.1.4	GDAL Grid Tutorial	911
8.1.4.1	Introduction to Gridding	911
8.1.4.2	Interpolation of the Scattered Data	912
8.1.4.3	Data Metrics Computation	913
8.1.4.4	Search Ellipse	914
8.2	Multidimensional raster	915
8.2.1	Multidimensional raster API tutorial	915
8.2.1.1	Read the content of an array	915
8.2.1.2	Other examples	918
8.3	Vector	918
8.3.1	Vector API tutorial	918
8.3.1.1	Reading From OGR	918
8.3.1.2	Writing To OGR	928
8.3.2	Vector driver implementation tutorial	938
8.3.2.1	Overall Approach	938
8.3.2.2	Implementing GDALDriver	938
8.3.2.3	Basic Read Only Data Source	939
8.3.2.4	Read Only Layer	941
8.3.3	Vector driver in Python implementation tutorial	944
8.3.3.1	Introduction	944
8.3.3.2	Linking mechanism to a Python interpreter	944
8.3.3.3	Driver location	944
8.3.3.4	Import section	944
8.3.3.5	Metadata section	945
8.3.3.6	Driver class	945
8.3.3.7	Dataset class	946
8.3.3.8	Layer class	947
8.3.3.9	Full example	950
8.3.3.10	Other examples	953
8.4	Geographic Network Model	953
8.4.1	GNM API tutorial	953
8.4.1.1	Managing networks	953
8.4.1.2	Analysing networks	958
8.5	Projections and Spatial Reference Systems tutorial (OSR - OGRSpatialReference)	960
8.5.1	OGR Coordinate Reference Systems and Coordinate Transformation tutorial	960
8.5.1.1	OGC WKT Coordinate System Issues	960
8.5.1.2	Introduction	965
8.5.1.3	References and applicable standards	965
8.5.1.4	Defining a Geographic Coordinate Reference System	965
8.5.1.5	CRS and axis order	968
8.5.1.6	Defining a Projected CRS	968
8.5.1.7	Querying Coordinate Reference System	969
8.5.1.8	Coordinate Transformation	970
8.5.1.9	Advanced Coordinate Transformation	971
8.5.1.10	Alternate Interfaces	972
8.5.1.11	History and implementation considerations	974
9	Community	975
9.1	Mailing List	975
9.2	GitHub	975
9.3	Chat	975
9.4	Conference	976
9.5	Governance and Community Participation	976

9.5.1	OSGeo Project Membership	976
9.5.2	Project Steering Committee	976
10	How to contribute?	979
10.1	Developer Contributions to GDAL	979
10.1.1	Git workflows with GDAL	979
10.1.1.1	Commit message	979
10.1.1.2	Initiate your work repository	980
10.1.1.3	Updating your local master against upstream master	980
10.1.1.4	Working with a feature branch	980
10.1.1.5	Backporting bugfixes from master to a stable branch	981
10.1.1.6	Things you should NOT do	981
10.2	Sphinx RST Style guide	981
10.2.1	Basic markup	981
10.2.2	Basic markup	981
10.2.3	Lists	981
10.2.4	List-tables	982
10.2.5	Page labels	982
10.2.6	Linking	983
10.2.7	Sections	983
10.2.8	Notes and warnings	983
10.2.9	Images	984
10.2.10	External files	984
10.2.11	Reference files and paths	985
10.2.12	Reference commands	986
11	FAQ	987
11.1	What does GDAL stand for?	987
11.2	What is this OGR stuff?	987
11.3	What does OGR stand for?	987
11.4	What does CPL stand for?	987
11.5	When was the GDAL project started?	988
11.6	Is GDAL/OGR proprietary software?	988
11.7	What license does GDAL/OGR use?	988
11.8	What operating systems does GDAL-OGR run on?	988
11.9	Is there a graphical user interface to GDAL/OGR?	988
11.9.1	Software using GDAL	988
11.10	What compiler can I use to build GDAL/OGR?	992
11.11	I have a question that's not answered here. Where can I get more information?	992
11.12	How do I add support for a new format?	992
11.13	How do I cite GDAL ?	992
12	License	993
12.1	License	993
	Bibliography	995
	Index	997

WHAT IS GDAL?

GDAL is a translator library for raster and vector geospatial data formats that is released under an X/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 January 2020 GDAL/OGR 3.0.4 release.



See *Software using GDAL*

DOWNLOAD

2.1 Current Releases

- **2020-01-28** gdal-3.0.4.tar.gz 3.0.4 Release Notes (3.0.4 md5)
- **2020-01-08** gdal-2.4.4.tar.gz 2.4.4 Release Notes (2.4.4 md5)

2.2 Past Releases

- **2020-01-08** gdal-3.0.3.tar.gz 3.0.3 Release Notes (3.0.3 md5)
- **2019-10-28** gdal-3.0.2.tar.gz 3.0.2 Release Notes (3.0.2 md5)
- **2019-10-28** gdal-2.4.3.tar.gz 2.4.3 Release Notes (2.4.3 md5)
- **2019-06-28** gdal-3.0.1.tar.gz 3.0.1 Release Notes (3.0.1 md5)
- **2019-06-28** gdal-2.4.2.tar.gz 2.4.2 Release Notes (2.4.2 md5)

2.3 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
```

2.4 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.

2.4.1 Windows

Windows builds are available via [Conda Forge](#) (64-bit only). See the [Conda](#) section for more detailed information.

2.4.2 Debian

Debian packages are now available on [Debian Unstable](#).

2.4.3 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 at <https://anaconda.org/conda-forge/gdal>.

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

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

2.4.4 Linux Docker images

Images with nightly builds of GDAL master and tagged releases are available at [Docker Hub](#)

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

PROGRAMS

3.1 Raster programs

3.1.1 Common options for raster programs

All GDAL command line programs support the following common options.

--version

Report the version of GDAL and exit.

--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 CreateCopy); 'rw+' is read, write and update (i.e. supports Create). A 'v' is appended for formats supporting virtual IO (/vsimem, /vsizip, etc). A 's' is appended for formats supporting subdatasets. Note: The valid formats for the output of gdalwarp are formats that support the Create() method (marked as rw+), not just the CreateCopy() method.

--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.

--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 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.

3.1.1.1 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 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.

New in version 2.3: If not specified, the format is guessed from the extension. Previously, it was generally GTiff for raster, or ESRI Shapefile for vector.

-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 `--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.

-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.

3.1.2 gdalinfo

3.1.2.1 Synopsis

```
gdalinfo [--help-general] [-json] [-mm] [-stats] [-hist] [-nogcp] [-nomd]
          [-norat] [-noct] [-nofl] [-checksum] [-proj4]
          [-listmdd] [-mdd domain\`all`] [-wkt_format WKT1|WKT2|...]
          [-sd subdataset] [-oo NAME=VALUE]* datasetname
```

3.1.2.2 Description

gdalinfo program lists various information about a GDAL supported raster dataset.

The following command line parameters can appear in any order

-json

Display the output in json format.

-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 want 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.

-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|WKT2|WKT2_2015|WKT2_2018

WKT format used to display the SRS. Currently the supported values are:

WKT1

WKT2 (latest WKT version, currently *WKT2_2018*)

WKT2_2015

WKT2_2018

New 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).

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.

3.1.2.3 C API

This utility is also callable from C with `GDALInfo()`.

New in version 2.1.

3.1.2.4 Example

```
gdalinfo ~/openev/utm.tif
Driver: GTiff/GeoTIFF
Size is 512, 512
Coordinate System is:
PROJCS["NAD27 / UTM zone 11N",
    GEOGCS["NAD27",
        DATUM["North_American_Datum_1927",
            SPHEROID["Clarke 1866",6378206.4,294.978698213901]],
        PRIMEM["Greenwich",0],
        UNIT["degree",0.0174532925199433]],
    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]]
Origin = (440720.000000,3751320.000000)
Pixel Size = (60.000000,-60.000000)
Corner Coordinates:
Upper Left ( 440720.000, 3751320.000) (117d38'28.21"W, 33d54'8.47"N)
Lower Left ( 440720.000, 3720600.000) (117d38'20.79"W, 33d37'31.04"N)
Upper Right ( 471440.000, 3751320.000) (117d18'32.07"W, 33d54'13.08"N)
Lower Right ( 471440.000, 3720600.000) (117d18'28.50"W, 33d37'35.61"N)
Center ( 456080.000, 3735960.000) (117d28'27.39"W, 33d45'52.46"N)
Band 1 Block=512x16 Type=Byte, ColorInterp=Gray
```

3.1.3 gdal_translate

3.1.3.1 Synopsis

```
gdal_translate [--help-general]
    [-ot {Byte/Int16/UInt16/UInt32/Int32/Float32/Float64/
        CInt16/CInt32/CFloat32/CFloat64}] [-strict]
    [-of format] [-b band]* [-mask band] [-expand {gray|rgb|rgba}]
    [-outsize xsize[%]|0 ysize[%]|0] [-tr xres yres]
    [-r {nearest,bilinear,cubic,cubicspline,lanczos,average,mode}]
    [-unscale] [-scale[_bn] [src_min src_max [dst_min dst_max]]]* [-exponent[_bn] exp_
    ↪val]*
    [-srcwin xoff yoff xsize ysize] [-epo] [-eco]
```

(continues on next page)

(continued from previous page)

```

[-projwin ulx uly lrx lry] [-projwin_srs srs_def]
[-a_srs srs_def] [-a_ullr ulx uly lrx lry] [-a_nodata value]
[-a_scale value] [-a_offset value]
[-nogcp] [-gcp pixel line easting northing [elevation]]*
|-colorinterp{_bn} {red|green|blue|alpha|gray|undefined}]
|-colorinterp {red|green|blue|alpha|gray|undefined},...]
[-mo "META-TAG=VALUE"]* [-q] [-sds]
[-co "NAME=VALUE"]* [-stats] [-norat]
[-oo NAME=VALUE]*
src_dataset dst_dataset

```

3.1.3.2 Description

The **gdal_translate** utility can be used to convert raster data between different formats, potentially performing some operations like subsettings, resampling, and rescaling pixels in the process.

-ot <type>

Force the output image bands to have a specific type. Use type names (i.e. Byte, Int16,...)

-strict

Don't be forgiving of mismatches and lost data when translating to the output format.

-of <format>

Select the output format. Starting with GDAL 2.3, if not specified, the format is guessed from the extension (previously was GTiff). 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.

-outsizes <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 **-outsizes** and **-a_ullr**.

-r {nearest (default), bilinear, cubic, cubicspline, lanczos, average, mode}

Select a resampling algorithm.

-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 0 to 255. If omitted the input range is automatically computed from the source data. `-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>

To 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.

-unscale

Apply the scale/offset metadata for the bands to convert scaled values to unscaled values. It is also often necessary to reset the output datatype with the `-ot` switch.

-srcwin <xoff> <yoff> <xsize> <ysize>

Selects a subwindow from the source image for copying based on pixel/line location.

-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`).

Note: In GDAL 2.1.0 and 2.1.1, using `-projwin` with coordinates not aligned with pixels will result in a sub-pixel shift. This has been corrected in later versions. When selecting non-nearest neighbour resampling, starting with GDAL 2.1.0, sub-pixel accuracy is however used to get better results.

-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.

<p>Warning: This does not cause reprojection of the dataset to the specified SRS.</p>
--

-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 behaviour is to accept such requests, when they were considered as an error before.

-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. 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_scale <value>

Set band scaling value(no modification of pixel values is done)

New in version 2.3.

-a_offset<value>

Set band offset value (no modification of pixel values is done)

New in version 2.3.

-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.

-a_nodata <value>

Assign a specified nodata value to output bands. It can be set to *<i>none</i>* 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>

Override the color interpretation of band X (where X is a valid band number, starting at 1)

New in version 2.3.

-colorinterp <red|green|blue|alpha|gray|undefined[,red|green|blue|alpha|gray|undefined]*>

Override the color interpretation of all specified bands. For example `-colorinterp red,green,blue,alpha` for a 4 band output dataset.

New in version 2.3.

-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 *-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.

-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.

-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.

3.1.3.3 C API

This utility is also callable from C with `GDALTranslate()`.

New in version 2.1.

3.1.3.4 Examples

```
gdal_translate -of GTiff -co "TILED=YES" utm.tif utm_tiled.tif
```

To create 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
```

To create a RGBA dataset from a RGB dataset with a mask

```
gdal_translate withmask.tif rgba.tif -b 1 -b 2 -b 3 -b mask
```

3.1.4 gdaladdo

3.1.4.1 Synopsis

```
gdaladdo [-r {nearest,average,gauss,cubic,cubicspline,lanczos,average_magphase,mode}]
          [-b band]* [-minsize val]
          [-ro] [-clean] [-oo NAME=VALUE]* [--help-general] filename [levels]
```

3.1.4.2 Description

The **gdaladdo** utility can be used to build or rebuild overview images for most supported file formats with one of several downsampling algorithms.

-r {nearest (default), average, gauss, cubic, cubicspline, lanczos, average_magphase, mode}
Select a resampling algorithm.

nearest applies a nearest neighbour (simple sampling) resampler

average computes the average of all non-NODATA contributing pixels.

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.

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.

New in version 2.3.
- <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.

New in version 2.3: 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.

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. 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*)

3.1.4.3 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.

For JPEG compressed external overviews, the JPEG quality can be set with “*-config JPEG_QUALITY_OVERVIEW value*”

For LZW or DEFLATE compressed external overviews, the predictor value can be set with “*-config PREDICTOR_OVERVIEW 11213*”

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 starting with GDAL 2.3.0 (previously was IF_NEEDED). The behaviour 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.

See the documentation of the *GTiff – GeoTIFF File Format* driver for further explanations on all those options.

3.1.4.4 C API

Functionality of this utility can be done from C with `GDALBuildOverviews()`.

3.1.4.5 Examples

Create overviews, embedded in the supplied TIFF file, with automatic computation of levels (GDAL 2.3 or later)

```
gdaladdo -r average abc.tif
```

Create overviews, embedded in the supplied TIFF file:

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

Create an external compressed GeoTIFF overview file from the ERDAS .IMG file:

```
gdaladdo -ro --config COMPRESS_OVERVIEW DEFLATE erdas.img 2 4 8 16
```

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
```

Create an Erdas Imagine format overviews for the indicated JPEG file:

```
gdaladdo --config USE_RRD YES airphoto.jpg 3 9 27 81
```

Create overviews for a specific subdataset, like 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
```

3.1.5 gdalwarp

3.1.5.1 Synopsis

```
gdalwarp [--help-general] [--formats]
  [-s_srs srs_def] [-t_srs srs_def] [-ct string] [-to "NAME=VALUE"]* [-novshiftgrid]
  [-order n | -tps | -rpc | -geoloc] [-et err_threshold]
  [-refine_gcps tolerance [minimum_gcps]]
  [-te xmin ymin xmax ymax] [-te_srs srs_def]
  [-tr xres yres] [-tap] [-ts width height]
  [-ovr level|AUTO|AUTO-n|NONE] [-wo "NAME=VALUE"] [-ot Byte/Int16/...] [-wt Byte/
→Int16]
  [-srcnodata "value [value...]"] [-dstnodata "value [value...]"]
  [-srcalpha|-nosrcalpha] [-dstalpha]
  [-r resampling_method] [-wm memory_in_mb] [-multi] [-q]
  [-cutline datasource] [-cl layer] [-cwhere expression]
  [-csql statement] [-cblend dist_in_pixels] [-crop_to_cutline]
  [-of format] [-co "NAME=VALUE"]* [-overwrite]
  [-nomd] [-cvm meta_conflict_value] [-setci] [-oo NAME=VALUE]*
  [-doo NAME=VALUE]*
  srcfile* dstfile
```

3.1.5.2 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.

-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. Starting with GDAL 2.2, 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.

-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. Starting with GDAL 2.2, 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.

-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.

New in version 3.0.

-to <NAME=VALUE>

Set a transformer option suitable to pass to `GDALCreateGenImgProjTransformer2()`.

-novshiftgrid

Disable the use of vertical datum shift grids when one of the source or target SRS has an explicit vertical datum, and the input dataset is a single band dataset.

New in version 2.2.

-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.

-et <err_threshold>

Error threshold for transformation approximation (in pixel units - defaults to 0.125, unless, starting with GDAL 2.1, the RPC_DEM warping 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. Not 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>

Set output file resolution (in target georeferenced units)

-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.

-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 type. Use type names (i.e. Byte, Int16,...)
- 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, 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.
 - `mode`: mode resampling, selects the value which appears most often of all the sampled points.
 - `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)
- 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. Use a value of `None` to ignore intrinsic nodata settings on the source dataset.
- 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)
New in version 2.2.
- 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. The value is interpreted as being in megabytes if the value is less than 10000. For values ≥ 10000 , this is interpreted as bytes.
- 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`
- q**
Be quiet.
- of** <format>
Select the output format. Starting with GDAL 2.3, if not specified, the format is guessed from the extension (previously was GTiff). 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 `-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.
- cutline** <datasource>
Enable use of a blend cutline from the name OGR support datasource.
- 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.
- 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)

New in version 2.1.

<srcfile>

The source file name(s).

<dstfile>

The destination file name.

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 must 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.

When doing vertical shift adjustments, the transformer option `-to ERROR_ON_MISSING_VERT_SHIFT=YES` can be used to error out as soon as a vertical shift value is missing (instead of 0 being used).

Starting with GDAL 3.1, it is possible to use as output format a driver that only supports the `CreateCopy` operation. This may internally imply creation of a temporary file.

3.1.5.3 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:

New in version 2.2.

```
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:

New in version 2.2.

```
gdalwarp -overwrite in_dem.tif out_dem.tif -s_srs EPSG:4326+5773 -t_srs EPSG:4979
```

3.1.5.4 See also

Wiki page discussing options and behaviours of gdalwarp

3.1.6 gdaltindex

3.1.6.1 Synopsis

```
gdaltindex [-f format] [-tileindex field_name] [-write_absolute_path]
           [-skip_different_projection] [-t_srs target_srs]
           [-src_srs_name field_name] [-src_srs_format [AUTO|WKT|EPSG|PROJ]]
           [-lyr_name name] index_file [gdal_file]*
```

3.1.6.2 Description

This program builds a shapefile 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.

-f <format>

The OGR format of the output tile index file. Starting with GDAL 2.3, if not specified, the format is guessed from the extension (previously was ESRI Shapefile).

-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 <type>

The format in which the SRS of each tile must be written. Types can be AUTO, WKT, EPSG, PROJ.

-lyr_name <name>

Layer name to create/append to in the output tile index file.

index_file

The name of the output file to create/append to. The default shapefile will be created if it doesn't already exist, otherwise it will append to the existing file.

<gdal_file>

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.

3.1.6.3 Examples

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
```

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
```

3.1.6.4 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.

3.1.7 gdalbuildvrt**3.1.7.1 Synopsis**

```
gdalbuildvrt [-tileindex field_name]
              [-resolution {highest|lowest|average|user}]
              [-te xmin ymin xmax ymax] [-tr xres yres] [-tap]
              [-separate] [-b band]* [-sd subdataset]
              [-allow_projection_difference] [-q]
              [-optim {[AUTO]/VECTOR/RASTER}]
              [-addalpha] [-hidenodata]
              [-srcnodata "value [value...]"] [-vrtnodata "value [value...]"]
              [-a_srs srs_def]
              [-r {nearest,bilinear,cubic,cubicspline,lanczos,average,mode}]
              [-oo NAME=VALUE]*
              [-input_file_list my_list.txt] [-overwrite] output.vrt [gdalfile]*
```

3.1.7.2 Description

This program builds a VRT (Virtual Dataset) that is a mosaic of the list of input GDAL datasets. 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, or it can be a MapServer tileindex (see ref `gdaltindex` utility). In the later case, all entries in the tile index will be added to the VRT.

With *-separate*, each files goes into a separate band in the VRT dataset. Otherwise, the files are considered as tiles of a larger mosaic and the VRT file has as many bands as one of 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 amount of checks to assure that all files that will be put in the resulting VRT have similar characteristics : number of bands, projection, color interpretation... If not, files that do not match the common characteristics will be skipped. (This is only true in the default mode, and not when using the `-separate` option)

If there is some amount of spatial overlapping between files, the order of files appearing in the list of source matter: 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 less prioritary 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.

-tileindex

Use the specified value as the tile index field, instead of the default value which is 'location'.

-resolution {highest|lowest|average|user}

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.

-tr <res> <yres>

Set target resolution. The values must be expressed in georeferenced units. Both must be positive values. Specifying those 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.

-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.

-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`.

-hiddenodata

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.

-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 vrt. Multiple `-b` switches may be used to select a set of input bands.

-sd <subdataset>

If the input dataset contains several subdatasets use a subdataset with the specified number (starting from 1).

This is an alternative of giving the full subdataset name as an input.

-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. 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. In that case, only the first band of each dataset will be placed into a new band. Contrary to the default mode, it is not required that all bands have the same datatype.

-allow_projection_difference

When this option is specified, the utility will accept to make a VRT even if the input datasets have not the same projection. Note: this does not mean that they will be reprojected. Their projection will just be ignored.

-optim { [AUTO] / VECTOR / RASTER }

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.

New in version 2.3.

-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 (default), bilinear, cubic, cubicspline, lanczos, average, mode}

Select a resampling algorithm.

-oo NAME=VALUE

Dataset open option (format specific)

New in version 2.2.

-input_file_list <mylist.txt>

To specify a text file with an input filename on each line

-q

To disable the progress bar on the console

-overwrite

Overwrite the VRT if it already exists.

3.1.7.3 Examples

- Make a virtual mosaic from all TIFF files contained in a directory :

```
gdalbuildvrt doq_index.vrt doq/*.tif
```

- Make a virtual mosaic from files whose name is specified in a text file :

```
gdalbuildvrt -input_file_list my_list.txt doq_index.vrt
```

- Make a RGB virtual mosaic from 3 single-band input files :

```
gdalbuildvrt -separate rgb.vrt red.tif green.tif blue.tif
```

- Make a virtual mosaic with blue background colour (RGB: 0 0 255) :

```
gdalbuildvrt -hidenodata -vrtnodata "0 0 255" doq_index.vrt doq/*.tif
```

3.1.8 gdal_contour

3.1.8.1 Synopsis

```
gdal_contour [-b <band>] [-a <attribute_name>] [-amin <attribute_name>] [-amax  
↪<attribute_name>]  
             [-3d] [-inodata]  
             [-snodata n] [-i <interval>]  
             [-f <formatname>] [[-dsco NAME=VALUE] ...] [[-lco NAME=VALUE] ...]  
             [-off <offset>] [-fl <level> <level>...] [-e <exp_base>]  
             [-nln <outlayername>] [-q] [-p]  
             <src_filename> <dst_filename>
```

3.1.8.2 Description

The **gdal_contour** generates a vector contour file from the input raster elevation model (DEM).

New in version 1.7.0: 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.

-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.

New in version 2.4.0.

-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.

New in version 2.4.0.

-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.

-snodata <value>

Input pixel value to treat as “nodata”.

-f <format>

Create output in a particular format.

New in version 2.3.0: 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.
- off** <offset>
Offset from zero relative to which to interpret intervals.
- fl** <level>
Name one or more “fixed levels” to extract.
- e** <base>
Generate levels on an exponential scale: $base^k$, for k an integer.
New in version 2.4.0.
- nln** <name>
Provide a name for the output vector layer. Defaults to “contour”.
- p**
Generate contour polygons rather than contour lines.
New in version 2.4.0.
- q**
Be quiet.

3.1.8.3 C API

Functionality of this utility can be done from C with `GDALContourGenerate()`.

3.1.8.4 Example

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.

```
gdal_contour -a elev dem.tif contour.shp -i 10.0
```

3.1.9 gdaldem

3.1.9.1 Synopsis

```
gdaldem <mode> <input> <output> <options>
```

Generate a shaded relief map from any GDAL-supported elevation raster:

```
gdaldem hillshade input_dem output_hillshade
    [-z ZFactor (default=1)] [-s scale* (default=1)]
    [-az Azimuth (default=315)] [-alt Altitude (default=45)]
    [-alg ZevenbergenThorne] [-combined | -multidirectional | -igor]
    [-compute_edges] [-b Band (default=1)] [-of format] [-co "NAME=VALUE"] * [-
↪q]
```

Generate a slope map from any GDAL-supported elevation raster:

```
gdaldem slope input_dem output_slope_map
    [-p use percent slope (default=degrees)] [-s scale* (default=1)]
    [-alg ZevenbergenThorne]
    [-compute_edges] [-b Band (default=1)] [-of format] [-co "NAME=VALUE"]* [-
↪q]
```

Generate an aspect map from any GDAL-supported elevation raster, 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 (default=1)] [-of format] [-co "NAME=VALUE"]* [-
↪q]
```

Generate a color relief map from any GDAL-supported elevation raster:

```
gdaldem color-relief input_dem color_text_file output_color_relief_map
    [-alpha] [-exact_color_entry | -nearest_color_entry]
    [-b Band (default=1)] [-of format] [-co "NAME=VALUE"]* [-q]
where color_text_file contains lines of the format "elevation_value red green blue"
```

Generate a Terrain Ruggedness Index (TRI) map from any GDAL-supported elevation raster:

```
gdaldem TRI input_dem output_TRI_map
    [-compute_edges] [-b Band (default=1)] [-of format] [-q]
```

Generate a Topographic Position Index (TPI) map from any GDAL-supported elevation raster:

```
gdaldem TPI input_dem output_TPI_map
    [-compute_edges] [-b Band (default=1)] [-of format] [-q]
```

Generate a roughness map from any GDAL-supported elevation raster:

```
gdaldem roughness input_dem output_roughness_map
    [-compute_edges] [-b Band (default=1)] [-of format] [-q]
```

3.1.9.2 Description

The **gdaldem** generally assumes that x, y and z units are identical. If x (east-west) and y (north-south) units are identical, but z (elevation) units are different, the scale (-s) option 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 to 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.

The following general options are available:

input_dem

The input DEM raster to be processed

output_XXX_map

The output raster produced

-of <format>

Select the output format.

New in version 2.3.0: 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

alg `ZevenbergenThorne`

Use Zevenbergen & Thorne formula, instead of Horn's formula, to compute slope & aspect. The literature suggests Zevenbergen & Thorne to be more suited to smooth landscapes, whereas Horn's formula to perform better on rougher terrain.

-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 *-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.

-q

Suppress progress monitor 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.

3.1.9.3 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 :

- z** <factor>
Vertical exaggeration used to pre-multiply the elevations
- s** <scale>
Ratio of vertical units to horizontal. 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)
- 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.

New in version 2.2.
- igor**
shading which tries to minimize effects on other map features beneath. Can't be used with -alt option.

New 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 :

- p**
If specified, the slope will be expressed as percent slope. Otherwise, it is expressed as degrees
- s**

Ratio of vertical units to horizontal. 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).

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 specific options are available :

-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 specific 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 is defined as the mean difference between a central pixel and 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.

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.

3.1.9.4 C API

This utility is also callable from C with `GDALDEMProcessing()`.

New in version 2.1.

3.1.9.5 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.

3.1.9.6 See also

Documentation of related GRASS utilities:

http://grass.osgeo.org/grass64/manuals/html64_user/r.slope.aspect.html

http://grass.osgeo.org/grass64/manuals/html64_user/r.shaded.relief.html

http://grass.osgeo.org/grass64/manuals/html64_user/r.colors.html

3.1.10 rgb2pct

3.1.10.1 Synopsis

```
rgb2pct.py [-n colors | -pct palette_file] [-of format] <source_file> <dest_file>
```

3.1.10.2 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.

-n <color>

Select the number of colors in the generated color table. Defaults to 256. Must be between 2 and 256.

-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 a raster file in a GDAL supported format with a palette.

-of <format>

Select the output format. Starting with GDAL 2.3, if not specified, the format is guessed from the extension (previously was GTiff). 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.

NOTE: `rgb2pct.py` is a Python script, and will only work if GDAL was built with Python support.

3.1.10.3 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.py -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>
```

3.1.11 pct2rgb

3.1.11.1 Synopsis

```
pct2rgb.py [-of format] [-b band] [-rgba] source_file dest_file
```

3.1.11.2 Description

This utility will convert a pseudo-color band on the input file into an output RGB file of the desired format.

-of *<format>*

Select the output format. Starting with GDAL 2.3, if not specified, the format is guessed from the extension (previously was GTiff). 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).

<source_file>

The input file.

<dest_file>

The output RGB file that will be created.

NOTE: pct2rgb.py is a Python script, and will only work if GDAL was built with Python support.

The ‘-expand rgb|rgba’ option of *gdal_translate* obsoletes that utility.

3.1.12 gdal_merge

3.1.12.1 Synopsis

```
gdal_merge.py [-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_files
```

3.1.12.2 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.

-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. Starting with GDAL 2.3, if not specified, the format is guessed from the extension (previously was GTiff). 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 *-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.

-ot <type>

Force the output image bands to have a specific type. Use type names (i.e. Byte, Int16,...)

-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.

-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.

-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.

Note: `gdal_merge.py` is a Python script, and will only work if GDAL was built with Python support.

3.1.12.3 Example

Create an image with the pixels in all bands initialized to 255.

```
% gdal_merge.py -init 255 -o out.tif in1.tif in2.tif
```

Create an RGB image that shows blue in pixels with no data. The first two bands will be initialized to 0 and the third band will be initialized to 255.

```
% gdal_merge.py -init "0 0 255" -o out.tif in1.tif in2.tif
```

3.1.13 gdal2tiles

3.1.13.1 Synopsis

```
gdal2tiles.py [-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] [--xyz]
              --tilesize=TILESIZE
              [-g googlekey] [-b bingkey] input_file [output_dir]
```

3.1.13.2 Description

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.

- p** <PROFILE>, **--profile**=<PROFILE>
Tile cutting profile (mercator, geodetic, raster) - default 'mercator' (Google Maps compatible).
- r** <RESAMPLING>, **--resampling**=<RESAMPLING>
Resampling method (average, near, bilinear, cubic, cubicspline, lanczos, antialias, 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 TSM
New in version 3.1.
- z** <ZOOM>, **--zoom**=<ZOOM>
Zoom levels to render (format: '2-5' or '10').
- e, --resume**
Resume mode. Generate only missing files.
- a** <NODATA>, **--srcnodata**=<NODATA>
NODATA transparency value to assign to the input data.
- v, --verbose**
Generate verbose output of tile generation.
- q, --quiet**
Disable messages and status to stdout
New in version 2.1.
- processes**=<NB_PROCESSES>
Number of processes to use for tiling.
New in version 2.3.
- tilesize**=<TILESIZE>
Pixel size of the tiles.
New in version 3.1.
- 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, 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 <http://code.google.com/apis/maps/signup.html>.
- b** <BINGKEY>, **--bingkey**=<BINGKEY>
Bing Maps API key from <https://www.bingmapsportal.com/>

Note: gdal2tiles.py is a Python script that needs to be run against Python GDAL binding.

3.1.13.3 Examples

Basic example:

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

3.1.14 gdal_rasterize

3.1.14.1 Synopsis

```
gdal_rasterize [-b band]* [-i] [-at]
    {[-burn value]* | [-a attribute_name] | [-3d]} [-add]
    [-l layername]* [-where expression] [-sql select_statement]
    [-dialect dialect] [-of format] [-a_srs srs_def] [-to NAME=VALUE]*
    [-co "NAME=VALUE"]* [-a_nodata value] [-init value]*
    [-te xmin ymin xmax ymax] [-tr xres yres] [-tap] [-ts width height]
    [-ot {Byte/Int16/UInt16/UInt32/Int32/Float32/Float64/
          CInt16/CInt32/CFloat32/CFloat64}]
    [-optim {[AUTO]/VECTOR/RASTER}] [-q]
    <src_datasource> <dst_filename>
```

3.1.14.2 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.

Note that on the fly reprojection of vector data to the coordinate system of the raster data is only supported since GDAL 2.1.0.

- 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.
- 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. Defaults to disabled for normal rendering rules.
- 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.
- 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.
- 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.

New in version 2.1.
- of** <format>
Select the output format. Starting with GDAL 2.3, if not specified, the format is guessed from the extension (previously was GTiff). 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.

New in version 2.3.

-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 *-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.

-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.

-ts <width> <height>

Set output file size in pixels and lines. Note that *-ts* cannot be used with *-tr*

-ot <type>

Force the output bands to be of the indicated data type. Defaults to Float64

-optim { [AUTO] /VECTOR/RASTER }

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.

New in version 2.3.

-q

Suppress progress monitor and other non-error output.

<src_datasource>

Any OGR supported readable datasource.

<dst_filename>

The GDAL supported output file. Must support update mode access. This file will be created (or overwritten if it already exists):option:-of, *-a_nodata*, *-init*, *-a_srs*, *-co*, *-te*, *-tr*, *-tap*, *-ts*, or *-ot* options are used.

The program create 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.

3.1.14.3 C API

This utility is also callable from C with `GDALRasterize()`.

New in version 2.1.

3.1.14.4 Example

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
```

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
```

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
```

3.1.15 gdaltransform

3.1.15.1 Synopsis

```
gdaltransform [--help-general]
  [-i] [-s_srs srs_def] [-t_srs srs_def] [-to "NAME=VALUE"]
  [-ct proj_string] [-order n] [-tps] [-rpc] [-geoloc]
  [-gcp pixel line easting northing [elevation]]* [-output_xy]
  [srcfile [dstfile]]
```

3.1.15.2 Description

The `gdaltransform` utility reprojects a list of coordinates into any supported projection, including GCP-based transformations.

-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 GCs (i.e. EPSG:4296), PROJ.4 declarations (as above), or the name of a .prj file containing well known text.

-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 GCs (i.e. EPSG:4296), PROJ.4 declarations (as above), or the name of a .prj file containing well known text.

-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.

New 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)
- output_xy**
Restrict output to “x y” instead of “x y z”
- <srcfile>**
File with source projection definition or GCP’s. If not given, source projection is read from the command-line *-s_srs* or *-gcp* parameters
- <dstfile>**
File 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.

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.

3.1.15.3 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:

```
244510.77404604 166154.532871342 -1046.79270555763
```

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 metre:

```
2.0000005420366 49.0000003766711 -0.0222802283242345
```

3.1.16 nearblack

3.1.16.1 Synopsis

```
nearblack [-of format] [-white | [-color c1,c2,c3...cn]*] [-near dist] [-nb non_black_
→pixels]
          [-setalpha] [-setmask] [-o outfile] [-q] [-co "NAME=VALUE"]* infile
```

3.1.16.2 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.

-o <outfile>

The name of the output file to be created.

-of <format>

Select the output format. Starting with GDAL 2.3, if not specified, the format is guessed from the extension (previously was ERDAS Imagine .img). 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 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.

-q

Suppress progress monitor and other non-error output.

<infile>

The input file. Any GDAL supported format, any number of bands, normally 8bit Byte bands.

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.

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.

3.1.16.3 C API

This utility is also callable from C with *GDALNearblack()*.

New in version 2.1.

3.1.17 gdal_retile

3.1.17.1 Synopsis

```
gdal_retile.py [-v] [-co NAME=VALUE]* [-of out_format] [-ps pixelWidth pixelHeight]
               [-overlap val_in_pixel]
               [-ot {Byte/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_files
```

3.1.17.2 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. Optionally pyramid levels are generated. 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

-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. Starting with GDAL 2.3, if not specified, the format is guessed from the extension (previously was GTiff). 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 *-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.

-ot <type>

Force the output image bands to have a specific type. Use type names (i.e. Byte, Int16,...)

-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

New in version 2.2.

-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: tile-name,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.

Note: gdal_retile.py is a Python script, and will only work if GDAL was built with Python support.

3.1.18 gdal_grid

3.1.18.1 Synopsis

```
gdal_grid [-ot {Byte/Int16/UInt16/UInt32/Int32/Float32/Float64/
CInt16/CInt32/CFloat32/CFloat64}]
[-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] [-outsize xsize ysize]
```

(continues on next page)

```
[-a algorithm[:parameter1=value1]*] [-q]


```

3.1.18.2 Description

This program creates 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/CPU's of the computer.

- ot** <type>
Force the output image bands to have a specific type. Use type names (i.e. Byte, Int16,...)
- of** <format>
Select the output format. Starting with GDAL 2.3, if not specified, the format is guessed from the extension (previously was GTiff). 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.
- outsize** <xsize ysize>
Set the size of the output file in pixels and lines.
- a_srs** <srs_def>
Override the projection for the output file. The <i>srs_def</i> 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 Z value read from feature geometry record (naturally, if you have a Z value in geometry, otherwise you have no choice and should specify a field name containing 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 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 Z field. This can be used for shift from e.g. foot to meters or from elevation to deep. The result value will be (Z value + Z increase value) * Z multiply value. The default value is 1.
- a** <[algorithm[:parameter1=value1][:parameter2=value2]...]>
Set the interpolation algorithm or data metric name and (optionally) its parameters. See [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 of 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 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.

-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 *-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.

-q
Suppress progress monitor and other non-error output.

<src_datasource>
Any OGR supported readable datasource.

<dst_filename>
The GDAL supported output file.

3.1.18.3 Interpolation algorithms

There are 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 default algorithm. It has 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 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.
- **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 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.
- `nodata`: NODATA marker to fill empty points (default 0.0).

invdistnn

New in version 2.1.

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.
- `nodata`: NODATA marker to fill empty points (default 0.0).

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.
- `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. Default is 0.
- `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.

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.
- **angle**: Angle of search ellipse rotation in degrees (counter clockwise, default 0.0).
- **nodata**: NODATA marker to fill empty points (default 0.0).

linear

New in version 2.1.

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).

3.1.18.4 Data metrics

Besides the interpolation functionality `ref 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.

- `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.
- `nodata`: NODATA marker to fill empty points (default 0.0).

3.1.18.5 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 `ref gdal_grid`. All you need is create a virtual dataset header (VRT) for you CSV file and use it as input datasource for `ref gdal_grid`. You can find details on VRT format at [VRT – Virtual Format](#) description page.

Here is a small example. Let we have a CSV file called `<i>dem.csv</i>` 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 above data we will create `<i>dem.vrt</i>` 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. Z value will be used for interpolation. Now you can use `<i>dem.vrt</i>` with all OGR programs (start with `ref ogrinfo` to test that everything works fine). The datasource will contain single layer called `<i>"dem"</i>` filled with point features constructed from values in CSV file. Using this technique you can handle CSV files with more than three columns, switch columns, etc.

If your CSV file does not contain column headers then it can be handled 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.

3.1.18.6 C API

This utility is also callable from C with `GDALGrid()`.

3.1.18.7 Examples

The following would create raster TIFF file from 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.

```
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
```

The next 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 `<i>"Elevation"</i>` field. The `GDAL_NUM_THREADS` is also set to parallelize the computation.

```
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
```

3.1.19 gdal_proximity

3.1.19.1 Synopsis

```
gdal_proximity.py <srcfile> <dstfile> [-srcband n] [-dstband n]
                    [-of format] [-co name=value]*
                    [-ot Byte/Int16/Int32/Float32/etc]
                    [-values n,n,n] [-distunits PIXEL/GEO]
                    [-maxdist n] [-nodata n] [-use_input_nodata YES/NO]
                    [-fixed-buf-val n]
```

3.1.19.2 Description

The `gdal_proximity.py` script 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.

<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. Starting with GDAL 2.3, if not specified, the format is guessed from the extension (previously was GTiff). 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 *-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.

-ot <type>

Force the output image bands to have a specific type. Use type names (i.e. Byte, Int16,...)

-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.

3.1.20 gdal_polygonize

3.1.20.1 Synopsis

```
gdal_polygonize.py [-8] [-nomask] [-mask filename] <raster_file> [-b band]
                  [-q] [-f ogr_format] <out_file> [layer] [fieldname]
```

3.1.20.2 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, defaulting to GML format.

The utility is based on the `:GDALPolygonize()` function which has additional details on the algorithm.

-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. Starting with GDAL 2.2, 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. Starting with GDAL 2.3, if not specified, the format is guessed from the extension (previously was GML). Use the short format name

<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.

3.1.21 gdal_sieve

3.1.21.1 Synopsis

```
gdal_sieve.py [-q] [-st threshold] [-4] [-8] [-o name=value]
               srcfile [-nomask] [-mask filename] [-of format] [dstfile]
```

3.1.21.2 Description

gdal_sieve.py script 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.

3.1.22 gdal_fillnodata

3.1.22.1 Synopsis

```
gdal_fillnodata.py [-q] [-md max_distance] [-si smooth_iterations]
                  [-o name=value] [-b band]
                  srcfile [-nomask] [-mask filename] [-of format] [dstfile]
```

3.1.22.2 Description

gdal_fillnodata.py script 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.

-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.

srcfile

The source raster file used to identify target pixels. Only one band is used.

-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).

dstfile

The new file to create with the interpolated result. If not provided, the source band is updated in place.

-of format

Select the output format. The default is *GTiff – GeoTIFF File Format*. Use the short format name.

3.1.23 gdallocationinfo

3.1.23.1 Synopsis

```
Usage: gdallocationinfo [--help-general] [-xml] [-lifonly] [-valonly]
                        [-b band]* [-overview overview_level]
                        [-l_srs srs_def] [-geoloc] [-wgs84]
                        [-oo NAME=VALUE]* srcfile [x y]
```

3.1.23.2 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.

-xml

The output report will be XML formatted for convenient post processing.

-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.

-b <band>

Selects a band to query. Multiple bands can be listed. By default all bands are queried.

-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)

<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 implement 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 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.

3.1.23.3 Examples

Simple example reporting on pixel (256,256) on the file `utm.tif`.

```
$ gdallocationinfo utm.tif 256 256
Report:
Location: (256P,256L)
Band 1:
    Value: 115
```

Query 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>
```

Reading location from stdin.

```
$ cat coordinates.txt
443020 3748359
441197 3749005
443852 3747743

$ cat coordinates.txt | gdallocationinfo -geoloc utmsmall.tif
Report:
  Location: (38P,49L)
  Band 1:
    Value: 214
Report:
  Location: (7P,38L)
  Band 1:
    Value: 107
Report:
  Location: (52P,59L)
  Band 1:
    Value: 148
```

3.1.24 gdalsrsinfo

3.1.24.1 Synopsis

```
Usage: gdalsrsinfo [--single-line] [-V] [-e] [-o <out_type>] <srs_def>
```

3.1.24.2 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)

--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_2018
- wkt2: Latest WKT2 version supported, currently wkt2_2018
- wkt2_2015: OGC WKT2:2015
- wkt2_2018: OGC 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)

3.1.24.3 Example

```
$ gdalsrsinfo "EPSG:4326"

PROJ.4 : '+proj=longlat +datum=WGS84 +no_defs '

OGC WKT :
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"]]
```

```
$ gdalsrsinfo -o proj4 osr/data/lcc_esri.prj
'+proj=lcc +lat_1=34.33333333333334 +lat_2=36.16666666666666 +lat_0=33.75 +lon_0=-79.
↪+x_0=609601.22 +y_0=0 +datum=NAD83 +units=m +no_defs '
\endverbatim
```

```
$ gdalsrsinfo -o proj4 landsat.tif
PROJ.4 : '+proj=utm +zone=19 +south +datum=WGS84 +units=m +no_defs '
```

```
$ gdalsrsinfo -o wkt "EPSG:32722"

PROJCRS["WGS 84 / UTM zone 22S",
  BASEGEOGCRS["WGS 84",
    DATUM["World Geodetic System 1984",
      ELLIPSOID["WGS 84",6378137,298.257223563,
        LENGTHUNIT["metre",1]]],
    PRIMEM["Greenwich",0,
      ANGLEUNIT["degree",0.0174532925199433]]],
  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],
```

(continues on next page)

(continued from previous page)

```

        LENGTHUNIT["metre",1]],
    AXIS["(N)",north,
        ORDER[2],
        LENGTHUNIT["metre",1]],
    USAGE[
        SCOPE["unknown"],
        AREA["World - S hemisphere - 54°W to 48°W - by country"],
        BBOX[-80,-54,0,-48]],
    ID["EPSG",32722]]

```

```

$ gdalsrsinfo -o wkt_all "EPSG:4322"
OGC WKT 1:
GEOGCS["WGS 72",
    DATUM["World_Geodetic_System_1972",
        SPHEROID["WGS 72",6378135,298.26,
            AUTHORITY["EPSG","7043"]],
        TOWGS84[0,0,4.5,0,0,0.554,0.2263],
        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 :
BOUNDCRS[
    SOURCECRS[
        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]],
            AREA["World"],
            BBOX[-90,-180,90,180],
            ID["EPSG",4322]]],
    TARGETCRS[
        GEODCRS["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["latitude",north,
                ORDER[1],
                ANGLEUNIT["degree",0.0174532925199433]],

```

(continues on next page)

(continued from previous page)

```

        AXIS["longitude",east,
            ORDER[2],
            ANGLEUNIT["degree",0.0174532925199433]],
        ID["EPSG",4326]]],
ABRIDGEDTRANSFORMATION["WGS 72 to WGS 84 (1)",
    METHOD["Position Vector transformation (geog2D domain)",
        ID["EPSG",9606]],
    PARAMETER["X-axis translation",0,
        ID["EPSG",8605]],
    PARAMETER["Y-axis translation",0,
        ID["EPSG",8606]],
    PARAMETER["Z-axis translation",4.5,
        ID["EPSG",8607]],
    PARAMETER["X-axis rotation",0,
        ID["EPSG",8608]],
    PARAMETER["Y-axis rotation",0,
        ID["EPSG",8609]],
    PARAMETER["Z-axis rotation",0.554,
        ID["EPSG",8610]],
    PARAMETER["Scale difference",1.0000002263,
        ID["EPSG",8611]],
    AREA["World"],
    BBOX[-90,-180,90,180],
    ID["EPSG",1237]]]

OGC WKT2:2018 :
BOUNDCRS[
    SOURCECRS[
        GEOGCRS["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]],
            USAGE[
                SCOPE["unknown"],
                AREA["World"],
                BBOX[-90,-180,90,180]],
            ID["EPSG",4322]]],
    TARGETCRS[
        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["latitude",north,
                ORDER[1],
                ANGLEUNIT["degree",0.0174532925199433]],

```

(continues on next page)

(continued from previous page)

```

        AXIS["longitude",east,
            ORDER[2],
            ANGLEUNIT["degree",0.0174532925199433]],
        ID["EPSG",4326]]],
ABRIDGEDTRANSFORMATION["WGS 72 to WGS 84 (1)",
    METHOD["Position Vector transformation (geog2D domain)",
        ID["EPSG",9606]],
    PARAMETER["X-axis translation",0,
        ID["EPSG",8605]],
    PARAMETER["Y-axis translation",0,
        ID["EPSG",8606]],
    PARAMETER["Z-axis translation",4.5,
        ID["EPSG",8607]],
    PARAMETER["X-axis rotation",0,
        ID["EPSG",8608]],
    PARAMETER["Y-axis rotation",0,
        ID["EPSG",8609]],
    PARAMETER["Z-axis rotation",0.554,
        ID["EPSG",8610]],
    PARAMETER["Scale difference",1.0000002263,
        ID["EPSG",8611]],
    USAGE[
        SCOPE["unknown"],
        AREA["World"],
        BBOX[-90,-180,90,180]],
    ID["EPSG",1237]]]

OGC WKT 1 (simple) :
GEOGCS["WGS 72",
    DATUM["World_Geodetic_System_1972",
        SPHEROID["WGS 72",6378135,298.26],
        TOWGS84[0,0,4.5,0,0,0.554,0.2263]],
    PRIMEM["Greenwich",0],
    UNIT["degree",0.0174532925199433]]

OGC WKT 1 (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]]

```

3.1.25 gdalmove

3.1.25.1 Synopsis

```
gdalmove.py [-s_srs <srs_defn>] -t_srs <srs_defn>
            [-et <max_pixel_err>] <target_file>
```

3.1.25.2 Description

The **gdalmove.py** 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.

-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.

3.1.26 gdal_edit

3.1.26.1 Synopsis

```
gdal_edit [--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]
          [-unsetstats] [-stats] [-approx_stats]
          [-setstats min max mean stddev]
          [-scale value] [-offset value] [-units value]
          [-colorinterp_X red|green|blue|alpha|gray|undefined]*
          [-gcp pixel line easting northing [elevation]]*
          [-unsetmd] [-oo NAME=VALUE]* [-mo "META-TAG=VALUE"]* datasetname
```

3.1.26.2 Description

The `gdal_edit.py` script 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).

--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.

New in version 1.11.

-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_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).

New 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.

New in version 2.4.

-unsetstats

Remove band statistics information.

New in version 2.0.

-stats

Calculate and store band statistics.

New in version 2.0.

-setstatsmin 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.

New in version 2.4.

-approx_stats

Calculate and store approximate band statistics.

New in version 2.0.

-a_nodata <value>

Assign a specified nodata value to output bands.

-unsetnodata

Remove existing nodata values.

New in version 2.1.

-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 is 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: $\text{true_value} = (\text{pixel_value} * \text{scale}) + \text{offset}$

Note: These values can be applied using `-unscale` during a `gdal_translate` run.

New in version 2.2.

-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 is recommended to set the value to 0. For more see scale.

New in version 2.2.

-units <value>

Assign a unit to output band(s).

New in version 3.1.

..`option:: colorinterp_X red|green|blue|alpha|gray|undefined`

Change the color interpretation of band X (where X is a valid band number, starting at 1).

New in version 2.3.

-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`.

New in version 2.0.

-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).

New in version 2.0.

The `-a_ullr`, `-a_ulurll`, `-tr` and `-unsetgt` options are exclusive.

The `-unsetstats` and either `-stats` or `-approx_stats` options are exclusive.

3.1.26.3 Example

```
gdal_edit -mo DATUM=WGS84 -mo PROJ=GEODETIC -a_ullr 7 47 8 46 test.ecw
```

```
gdal_edit -scale 1e3 1e4 -offset 0 10 twoBand.tif
```

3.1.27 gdal_calc.py

3.1.27.1 Synopsis

```
gdal_calc.py --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, but no projection checking is performed.

--help

Show this help message and exit

-h

The same as `--help`.

--calc=expression

Calculation in gdalnumeric syntax using `+`, `-`, `/`, `*`, or any numpy array functions (i.e. `log10()`).

-A <filename>

Input gdal raster file, you can use any letter (A-Z).

--A_band=<n>

Number of raster band for file A (default 1).

--outfile=<filename>

Output file to generate or fill.

--NoDataValue=<value>

Output nodata value (default datatype specific value).

--type=<datatype>

Output datatype, must be one of [Int32, Int16, Float64, UInt16, Byte, UInt32, Float32].

--format=<gdal_format>

GDAL format for output file.

--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]
Process all bands of given raster (A-Z).

--overwrite
Overwrite output file if it already exists.

--debug
Print debugging information.

--quiet
Suppress progress messages.

3.1.27.2 Example

Add two files together:

```
gdal_calc.py -A input1.tif -B input2.tif --outfile=result.tif --calc="A+B"
```

Average of two layers:

```
gdal_calc.py -A input.tif -B input2.tif --outfile=result.tif --calc="(A+B)/2"
```

Set values of zero and below to null:

```
gdal_calc.py -A input.tif --outfile=result.tif --calc="A*(A>0)" --NoDataValue=0
```

3.1.28 gdal_pansharpen.py

3.1.28.1 Synopsis

```
gdal_pansharpen [--help-general] pan_dataset {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]
```

3.1.28.2 Description

The **gdal_pansharpen.py** script 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.

-of <format>:
Select the output format. Starting with GDAL 2.3, if not specified, the format is guessed from the extension (previously was `GTiff`). 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 (default),cubicspline,lanczos,average}
Select a resampling algorithm.

-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(default),intersection,none,nonewithoutwarning}
Select behaviour when bands have not the same extent. See *SpatialExtentAdjustment* documentation in [VRT tutorial](#)

-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 *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.

-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.

3.1.28.3 Example

With spectral bands in a single dataset :

```
gdal_pansharpen.py panchro.tif multispectral.tif pansharpened_out.tif
```

With a few spectral bands from a single dataset, reordered :

```
gdal_pansharpen.py panchro.tif multispectral.tif,band=3 multispectral.tif,band=2 ↵
↵multispectral.tif,band=1 pansharpened_out.tif
```

With spectral bands in several datasets :

```
gdal_pansharpen.py panchro.tif band1.tif band2.tif band3.tif pansharpened_out.tif
```

Specify weights:

```
gdal_pansharpen.py -w 0.7 -w 0.2 -w 0.1 panchro.tif multispectral.tif pansharpened_
→out.tif
```

Specify RGB bands from a RGBNir multispectral dataset while computing the pseudo panchromatic intensity on the 4 RGBNir bands:

```
gdal_pansharpen.py -b 1 -b 2 -b 3 panchro.tif rgbnir.tif pansharpened_out.tif
```

3.1.29 gdal-config

Determines various information about a GDAL installation.

3.1.29.1 Synopsis

```
gdal-config [OPTIONS]
Options:
    [--prefix[=DIR]]
    [--libs]
    [--cflags]
    [--version]
    [--ogr-enabled]
    [--formats]
```

3.1.29.2 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.

3.1.30 gdalmanage

3.1.30.1 Synopsis

```
Usage: gdalmanage mode [-r] [-u] [-f format]
                        datasetname [newdatasetname]
```

3.1.30.2 Description

The **gdalmanage** program can perform various operations on raster data files, depending on the chosen *mode*. This includes identifying raster data types and deleting, renaming or copying the files.

<mode>

Mode of operation

identify `` `` <datasetname>: List data format of file.

copy `` `` <datasetname> `` `` <newdatasetname>: Create a copy of the raster file with a new name.

rename `` `` <datasetname> `` `` <newdatasetname>: Change the name of the raster file.

delete `` `` <datasetname>: Delete raster file.

-r

Recursively scan files/folders for raster files.

-u

Report failures if file type is unidentified.

-f <format>

Specify format of raster file if unknown by the application. Uses short data format name (e.g. *GTiff*).

<datasetname>

Raster file to operate on.

<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.

3.1.30.3 Examples

Using identify mode

Report the data format of the raster 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
```

(continues on next page)

(continued from previous page)

```
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 raster data:

```
$ gdalmanage copy NE1_50M_SR_W.tif nel_copy.tif
```

Using rename mode

Rename raster data:

```
$ gdalmanage rename NE1_50M_SR_W.tif nel_rename.tif
```

Using delete mode

Delete the raster data:

```
gdalmanage delete NE1_50M_SR_W.tif
```

3.1.31 gdalcompare

3.1.31.1 Synopsis

```
gdalcompare.py [-sds] golden_file new_file
```

3.1.31.2 Description

The **gdalcompare.py** script 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.

-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.py` script can also be called as a library from python code though it is not typically in the python path for including. The primary entry point is `gdalcompare.compare()` 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.

3.1.32 gdal_viewshed

New in version 3.1.0.

3.1.32.1 Synopsis

```
gdal_viewshed [-b <band>]
               [-a_nodata <value>] [-f <formatname>]
               [-oz <observer_height>] [-tz <target_height>] [-md <max_distance>]
               -ox <observer_x> -oy <observer_y>
               [-vv <visibility>] [-iv <invisibility>]
               [-ov <out_of_range>] [-cc <curvature_coef>]
               [[-co NAME=VALUE] ...]
               [-q] [-om <output mode>]
               <src_filename> <dst_filename>
```

3.1.32.2 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.

-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 `-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.

-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).
- oy** <value>
The Y position of the observer (in SRS units).
- 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.
- cc** <value>
Coefficient to consider the effect of the curvature and refraction. The height of the DEM is corrected according to the following formula:

$$Height_{Corrected} = Height_{DEM} - CurvCoef \frac{TargetDistance^2}{SphereDiameter}$$

For atmospheric refraction we can use 0.85714

- iv** <value>
Pixel value to set for invisible areas. 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. Default: 0
- vv** <value>
Pixel value to set for visible areas. Default: 255
- om** <output mode>
Sets what information the output contains.
Possible values: VISIBLE, DEM, GROUND
VISIBLE 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. Flags -tz, -iv and -vv will be ignored.
Default VISIBLE

3.1.32.3 C API

Functionality of this utility can be done from C with `GDALViewshedGenerate()`.

3.1.32.4 Example

Compute the visibility of an elevation raster data source with defaults

```
gdal_viewshed -md 500 -ox -10147017 -oy 5108065 source.tif destination.tif
```

- *Common options*
- *gdalinfo*: Lists information about a raster dataset.
- *gdal_translate*: Converts raster data between different formats.

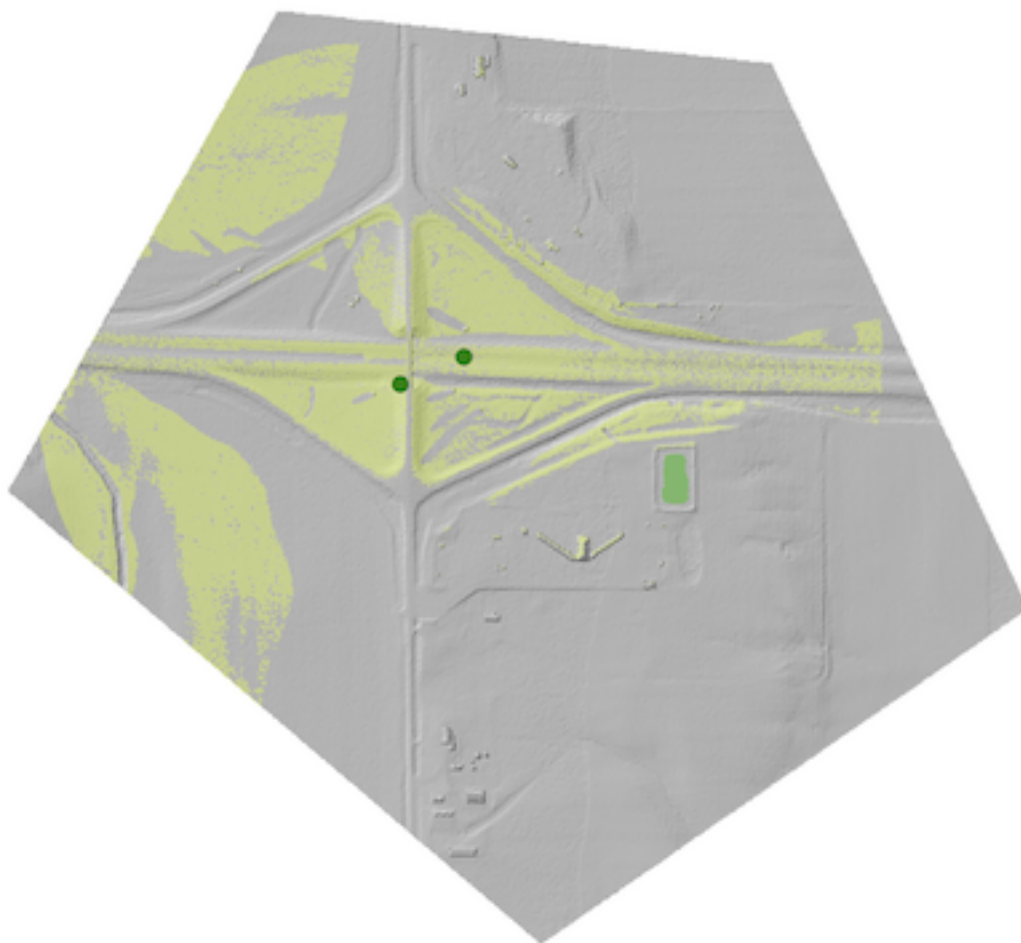


Fig. 1: A computed visibility for two separate $-ox$ and $-oy$ points on a DEM.

- *gdaladdo*: Builds or rebuilds overview images.
- *gdalwarp*: Image reprojection and warping utility.
- *gdaltindex*: Builds a shapefile as a raster tileindex.
- *gdalbuildvrt*: Builds a VRT from a list of datasets.
- *gdal_contour*: Builds vector contour lines from a raster elevation model.
- *gdaldem*: Tools to analyze and visualize DEMs.
- *rgb2pct*: Convert a 24bit RGB image to 8bit paletted.
- *pct2rgb*: Convert an 8bit paletted image to 24bit RGB.
- *gdal_merge*: Mosaics a set of images.
- *gdal2tiles*: Generates directory with TMS tiles, KMLs and simple web viewers.
- *gdal_rasterize*: Burns vector geometries into a raster.
- *gdaltransform*: Transforms coordinates.
- *nearblack*: Convert nearly black/white borders to black.
- *gdal_retile*: Retiles a set of tiles and/or build tiled pyramid levels.
- *gdal_grid*: Creates regular grid from the scattered data.
- *gdal_proximity*: Produces a raster proximity map.
- *gdal_polygonize*: Produces a polygon feature layer from a raster.
- *gdal_sieve*: Removes small raster polygons.
- *gdal_fillnodata*: Fill raster regions by interpolation from edges.
- *gdallocationinfo*: Raster query tool
- *gdalsrsinfo*: Lists info about a given SRS in number of formats (WKT, PROJ.4, etc.)
- *gdalmove*: Transform georeferencing of raster file in place.
- *gdal_edit*: Edit in place various information of an existing GDAL dataset.
- *gdal_calc.py*: Command line raster calculator with numpy syntax.
- *gdal_pansharpen.py*: Perform a pansharpen operation.
- *gdal-config*: Determines various information about a GDAL installation.
- *gdalmanage*: Identify, delete, rename and copy raster data files.
- *gdalcompare*: Compare two images.
- *gdal_viewshed*: Compute a visibility mask for a raster.

3.2 Multidimensional Raster programs

3.2.1 gdalmdiminfo

New in version 3.1.

3.2.1.1 Synopsis

```
gdalmdiminfo [--help-general] [-oo NAME=VALUE]* [-arrayoption NAME=VALUE]*
              [-detailed] [-noproretty] [-array {array_name}] [-limit {number}]
              <datasetname>
```

3.2.1.2 Description

gdalinfo program lists various information about a GDAL supported multidimensional raster dataset as JSON output. It follows the following [JSON schema](#)

The following command line parameters can appear in any order

-detailed

Most verbose output. Report attribute data types and array values.

-noproretty

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.

3.2.1.3 C API

This utility is also callable from C with `GDALMultiDimInfo()`.

3.2.1.4 Examples

- Display general structure1

```
$ gdalmdiminfo netcdf-4d.nc
```

```
{
  "type": "group",
  "name": "/",
  "attributes": {
```

(continues on next page)

(continued from previous page)

```

    "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": [
        "/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"
    }
  },

```

(continues on next page)

(continued from previous page)

```

"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
}
},
"structural_info": {
  "NC_FORMAT": "CLASSIC"
}
}

```

- Display detailed information about a given array

```
$ gdalmdiminfo netcdf-4d.nc -array t -detailed -limit 3
```

3.2.2 gdalmdimtranslate

New in version 3.1.

3.2.2.1 Synopsis

```
gdalmdimtranslate [--help-general] [-co "NAME=VALUE"]*
                  [-of format] [-array <array_spec>]*
                  [-group <group_spec>]*
                  [-subset <subset_spec>]*
                  [-scaleaxes <scaleaxes_spec>]*
                  <src_filename> <dst_filename>
```

3.2.2.2 Description

gdalmdimtranslate program converts multidimensional raster between different formats, and/or can perform selective conversion of specific arrays and groups, and/or subsetting operations.

The following command line parameters can appear in any order.

-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. 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 *-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.

-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}][,transpose=[{axis1},{axis2},...][,view={view_expr}]

[{axis1},{axis2},...] is the argument of *GDALMDArray::Transpose()*. For example, transpose=[1,0] switches the axis order of a 2D array.

{view_expr} is the value of the *viewExpr* argument of *GDALMDArray::GetView()*

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.

-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 of specifying a *view* option in `-array`.

-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 of specifying a *view* option in `-array`.

<src_dataset>

The source dataset name.

<dst_dataset>

The destination file name.

3.2.2.3 C API

This utility is also callable from C with `GDALMultiDimTranslate()`.

3.2.2.4 Examples

- Convert a netCDF file to a multidimensional VRT file

```
$ gdalmdimtranslate in.nc out.vrt
```

- Extract a 2D slice of a time,Y,X array

```
$ gdalmdimtranslate in.nc out.tif -subset 'time("2010-01-01")' -array temperature
```

- Subsample along X and Y axis

```
$ gdalmdimtranslate in.nc out.nc -scaleaxes "X(2),Y(2)"
```

- Reorder the values of a 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,:]"
```

- 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]"
```

- *gdalmdiminfo*: Reports structure and content of a multidimensional dataset.
- *gdalmdimtranslate*: Converts multidimensional data between different formats, and perform subsetting.

3.3 Vector programs

3.3.1 Common options for vector programs

All GDAL OGR command line programs support the following common options.

--version

Report the version of GDAL and exit.

--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 `CreateCopy()`); *rw+* is read, write and update (i.e. supports `Create`). A *v* is appended for formats supporting virtual IO (*/vsimem*, */vsigzip*, */vsizip*, etc). A *s* is appended for formats supporting subdatasets.

--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*.

--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_TEMPDIR* (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. 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 OGR command line options and exit.

3.3.2 ogrinfo

3.3.2.1 Synopsis

```
ogrinfo [--help-general] [-ro] [-q] [-where restricted_where|\\@filename]
        [-spat xmin ymin xmax ymax] [-geomfield field] [-fid fid]
        [-sql statement|\\@filename] [-dialect dialect] [-al] [-rl] [-so] [-fields=
→{YES/NO}]
        [-geom={YES/NO/SUMMARY/WKT/ISO_WKT}] [--formats] [[-oo NAME=VALUE] ...]
        [-nomd] [-listmdd] [-mdd domain|`all`]*
        [-nocount] [-noextent] [-nogeomtype] [-wkt_format WKT1|WKT2|...]
        <datasource_name> [<layer> [<layer> ...]]
```

3.3.2.2 Description

The **ogrinfo** program lists various information about an OGR-supported data source to stdout (the terminal).

- ro**
Open the data source in read-only mode.
- al**
List all features of all layers (used instead of having to give layer names as arguments).
- 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).
New in version 2.2.
- so**
Summary Only: suppress listing of individual features and show only summary information like projection, schema, feature count and extents.
- 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. Starting with GDAL 2.1, the \filename syntax can be used to indicate that the content is in the pointed filename.
- sql** <statement>
Execute the indicated SQL statement and return the result. Starting with GDAL 2.1, the @filename syntax can be used to indicate that the content is in the pointed filename.
- 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.
- 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.
- 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. (WKT and ISO_WKT are available starting with GDAL 2.1, which also changes the default to ISO_WKT)
- oo** NAME=VALUE
Dataset open option (format-specific)
- 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.

-nogeomtype

Suppress layer geometry type printing.

New in version 3.1.

--formats

List the format drivers that are enabled.

-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

New 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.

3.3.2.3 Example

Example reporting all layers in an NTF file:

```
ogrinfo wrk/SKETLAND_ISLANDS.NTF

# INFO: Open of `wrk/SKETLAND_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 of retrieving a summary (`-so`) of a layer without showing details about every single 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",
#       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 using an attribute query is used to restrict the output of the features in a layer:

```
ogrinfo -ro \
-where 'GLOBAL_LINK_ID=185878' \
wrk/SHETLAND_ISLANDS.NTF BL2000_LINK

# INFO: Open of `wrk/SHETLAND_ISLANDS.NTF'
# using driver `UK .NTF' successful.
#
# Layer name: BL2000_LINK
# Geometry: Line String
# Feature Count: 1
# Extent: (419794.100000, 1069031.000000) - (419927.900000, 1069153.500000)
# Layer SRS WKT:
# PROJCS["OSGB 1936 / British National Grid",
#   GEOGCS["OSGB 1936",
#     DATUM["OSGB_1936",
#       SPHEROID["Airy 1830",6377563.396,299.3249646]],
#     PRIMEM["Greenwich",0],
#     UNIT["degree",0.0174532925199433]],
#   PROJECTION["Transverse_Mercator"],
#   PARAMETER["latitude_of_origin",49],
#   PARAMETER["central_meridian",-2],
#   PARAMETER["scale_factor",0.999601272],
#   PARAMETER["false_easting",400000],
#   PARAMETER["false_northing",-100000],
#   UNIT["metre",1]]
# LINE_ID: Integer (6.0)
```

(continues on next page)

(continued from previous page)

```
# GEOM_ID: Integer (6.0)
# FEAT_CODE: String (4.0)
# GLOBAL_LINK_ID: Integer (10.0)
# TILE_REF: String (10.0)
# 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 1069043.800,419808.300
# 1069048.800,419805.100 1069046.000,419805.000 1069040.600,419809.400
# 1069037.400,419827.400 1069035.600,419842 1069031,419859.000
# 1069032.800,419879.500 1069049.500,419886.700 1069061.400,419890.100
# 1069070.500,419890.900 1069081.800,419896.500 1069086.800,419898.400
# 1069092.900,419896.700 1069094.800,419892.500 1069094.300,419878.100
# 1069085.600,419875.400 1069087.300,419875.100 1069091.100,419872.200
# 1069094.600,419890.400 1069106.400,419907.600 1069112.800,419924.600
# 1069133.800,419927.900 1069146.300,419927.600 1069152.400,419922.600
# 1069153.500,419917.100 1069153.500,419911.500 1069153.000,419908.700
# 1069152.500,419903.400 1069150.800,419898.800 1069149.400,419894.800
# 1069149.300,419890.700 1069149.400,419890.600 1069149.400,419880.800
# 1069149.800,419876.900 1069148.900,419873.100 1069147.500,419870.200
# 1069146.400,419862.100 1069143.000,419860 1069142,419854.900
# 1069138.600,419850 1069135,419848.800 1069134.100,419843
# 1069130,419836.200 1069127.600,419824.600 1069123.800,419820.200
# 1069126.900,419815.500 1069126.900,419808.200 1069116.500,419798.700
# 1069117.600,419794.100 1069115.100,419796.300 1069109.100,419801.800
# 1069106.800,419805.000 1069107.300)
```

3.3.3 ogr2ogr

3.3.3.1 Synopsis

```
ogr2ogr [--help-general] [-skipfailures] [-append] [-update]
[-select field_list] [-where restricted_where|\@filename]
[-progress] [-sql <sql statement>|\@filename] [-dialect dialect]
[-preserve_fid] [-fid FID] [-limit nb_features]
[-spat xmin ymin xmax ymax] [-spat_srs srs_def] [-geomfield field]
[-a_srs srs_def] [-t_srs srs_def] [-s_srs srs_def] [-ct string]
[-f format_name] [-overwrite] [[-dsco NAME=VALUE] ...]
dst_datasource_name src_datasource_name
[-lco NAME=VALUE] [-nln name]
[-nlt type|PROMOTE_TO_MULTI|CONVERT_TO_LINEAR|CONVERT_TO_CURVE]
[-dim XY|XYZ|XYM|XYZM|2|3|layer_dim] [layer [layer ...]]

# Advanced options
[-gt n]
[[-oo NAME=VALUE] ...] [[-doo NAME=VALUE] ...]
[-clipsrc [xmin ymin xmax ymax]|WKT|datasource|spat_extent]
[-clipsrcsql sql_statement] [-clipsrclayer layer]
[-clipsrcwhere expression]
[-clipdst [xmin ymin xmax ymax]|WKT|datasource]
[-clipdstsql sql_statement] [-clipdstlayer layer]
```

(continues on next page)

(continued from previous page)

```

[-clipdstwhere expression]
[-wrapdateline] [-datelineoffset val]
[[-simplify tolerance] | [-segmentize max_dist]]
[-makevalid]
[-addfields] [-unsetFid]
[-relaxedFieldNameMatch] [-forceNullable] [-unsetDefault]
[-fieldTypeToString All|(type1[,type2]*)] [-unsetFieldWidth]
[-mapFieldType type1|All=type2[,type3=type4]*]
[-fieldmap identity | index1[,index2]*]
[-splitlistfields] [-maxsubfields val]
[-explodecollections] [-zfield field_name]
[-gcp ungeoref_x ungeoref_y georef_x georef_y [elevation]]* [-order n | -tps]
[-nomd] [-mo "META-TAG=VALUE"]* [-noNativeData]

```

3.3.3.2 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.

-f <format_name>

Output file format name, e.g. ESRI Shapefile, MapInfo File, PostgreSQL. Starting with GDAL 2.3, if not specified, the format is guessed from the extension (previously was ESRI Shapefile).

-append

Append to existing layer instead of creating new

-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. A field is skipped if mentioned previously in the list even if the input layer has duplicate field names. (Defaults to all; any field is skipped if a subsequent field with same name is found.) Geometry fields can also be specified in the list.

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>

SQL statement to execute. The resulting table/layer will be saved to the output. Starting with GDAL 2.1, the @filename syntax can be used to indicate that the content is in the pointed filename.

-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.

-where restricted_where

Attribute query (like SQL WHERE). Starting with GDAL 2.1, the @filename syntax can be used to indicate that the content is in the pointed filename.

-skipfailures

Continue after a failure, skipping the failed feature.

- 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.
- 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_MULTII 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 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_MULTII` 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. Support for M was added in GDAL 2.1.
- a_srs** <srs_def>
Assign an output SRS. Srs_def can be a full WKT definition (hard to escape properly), or a well known definition (i.e. EPSG:4326) or a file with a WKT definition.
- t_srs** <srs_def>
Reproject/transform to this SRS on output.
- s_srs** <srs_def>
Override source SRS.
- 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.

New in version 3.0.

-preserve_fid

Use the FID of the source features instead of letting the output driver automatically assign a new one (for formats that require an FID). If not in append mode, this behaviour 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 behaviour can be disabled by setting `-unsetFid`.

-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

Input dataset open option (format specific).

-doo NAME=VALUE

Destination dataset open option (format specific), only valid in -update mode.

-gt n

Group n features per transaction (default 20000). 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 to the specified bounding box (expressed in source SRS), WKT geometry (POLYGON or MULTIPOLYGON), from a datasource or to the spatial extent of the -spa.. option if you use the spat_extent keyword. When specifying a datasource, you will generally want to use it in combination of 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 attribute query.

-clipdst <xmin> <ymin> <xmax> <ymax>

Clip geometries after reprojection to the specified bounding box (expressed in dest SRS), WKT geometry (POLYGON or MULTIPOLYGON) or from a datasource. When specifying a datasource, you will generally want to use it in combination of the -clipdstlayer, -clipdstwhere or -clipdstsql.. options

-clipdstsql <sql_statement>

Select desired geometries using an SQL query instead.

-clipdstlayer <layername>

Select the named layer from the destination clip datasource.

-clipdstwhere <expression>

Restrict desired geometries based on 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.

-segmentize <max_dist>

Maximum distance between 2 nodes. Used to create intermediate points.

-makevalid

Run the *OGRGeometry::MakeValid()* operation, followed by *OGRGeometryFactory::removeLowerDimensions* on geometries to ensure they are valid regarding the rules of the Simple Features specification.

-fieldTypeToString type1,...

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.

-mapFieldType srctype|All=dsttype,...

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.

-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.

-zfield <field_name>

Uses the specified field to fill the Z coordinate of geometries.

-gcp <ungeoref_x> <ungeoref_y> <georef_x> <georef_y> <elevation>

Add the indicated ground control point. This option may be provided multiple times to provide a set of GCPs.

-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 specify to prevent the name of the source FID column and source feature IDs from being re-used for the target layer. This option can for example be useful if selecting source features with a `ORDER BY` clause.

-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.

New in version 2.1.

3.3.3.3 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.

3.3.3.4 C API

This utility is also callable from C with `GDALVectorTranslate()`.

3.3.3.5 Examples

Basic conversion from Shapefile to GeoPackage:

```
ogr2ogr \
  -f GPKG output.gpkg \
  input.shp
```

Change the coordinate reference system from EPSG:4326 to EPSG:3857:

```
ogr2ogr \
  -s_srs EPSG:4326 \
  -t_srs EPSG:3857 \
  -f GPKG output.gpkg \
  input.gpkg
```

Example appending to an existing layer (both `-update` and `-append` flags need to be used):

```
ogr2ogr -update -append -f PostgreSQL PG:dbname=warmerda abc.tab
```

Clip input layer with a bounding box (`<xmin> <ymin> <xmax> <ymax>`):

```
ogr2ogr \
  -spat -13.931 34.886 46.23 74.12 \
  -f GPKG output.gpkg \
  natural_earth_vector.gpkg
```

Filter Features by a `-where` clause:

```
ogr2ogr \
  -where "\"POP_EST\" < 1000000" \
  -f GPKG output.gpkg \
  natural_earth_vector.gpkg \
  ne_10m_admin_0_countries
```

Example 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 for using the `-fieldmap` setting. The first field of the source layer is used to fill the third field (index 2 = third field) of the target layer, the second field of the source layer is ignored, the third field of the source layer used to fill the fifth field of the target layer.

```
ogr2ogr -append -fieldmap 2,-1,4 dst.shp src.shp
```

More examples are given in the individual format pages.

3.3.4 ogrtindex

3.3.4.1 Synopsis

```
ogrindex [-lnum n]... [-lname name]... [-f output_format]
          [-write_absolute_path] [-skip_different_projection]
          [-t_srs target_srs]
          [-src_srs_name field_name] [-src_srs_format [AUTO|WKT|EPSG|PROJ]]
          [-accept_different_schemas]
          <output_dataset> <src_dataset>...
```

3.3.4.2 Description

ogrindex 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.

-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.

New in version 2.2.0.

-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.

New in version 2.2.0.

-src_srs_format <format>

The format in which the SRS of each tile must be written. Available formats are: AUTO, WKT, EPSG, PROJ.

New in version 2.2.0.

-accept_different_schemas

By default ogrindex 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.

3.3.4.3 Example

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 1069148.900,419873.100 1069147.500,419870.200
1069146.400,419862.100 1069143.000,419860 1069142,419854.900
1069138.600,419850 1069135,419848.800 1069134.100,419843
1069130,419836.200 1069127.600,419824.600 1069123.800,419820.200
1069126.900,419815.500 1069126.900,419808.200 1069116.500,419798.700
1069117.600,419794.100 1069115.100,419796.300 1069109.100,419801.800
1069106.800,419805.000 1069107.300)
```

3.3.5 ogrlineref

3.3.5.1 Synopsis

```
ogrlineref [--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] [-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]
```

3.3.5.2 Description

The **ogrlineref** program can be used for:

- create 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** creates a linear reference - a file containing a segments of special length (e.g. 1 km in reference units) and get coordinates, linear referenced distances or sublines (subpaths) from this file. The utility not required the M or Z values in geometry. The results can be stored in any OGR supported format. Also some information is written to the stdout.

--help-general

Show the usage.

-progress

Show progress.

-quiet

Suppress all messages except errors and results.

-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

3.3.5.3 Example

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
```

3.3.6 ogrmerge

3.3.6.1 Synopsis

```
ogrmerge.py -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]
```

3.3.6.2 Description

New in version 2.2.

ogrmerge.py script takes as input several vector datasets, each of them having one or several vector layers, and copy them in a target dataset.

There are essential 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 – 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.

-
- o** <out_dsname>
Output dataset name. Required.
- <src_dsname>**
One or several input vector datasets. Required
- f** <format>
Select the output format. Starting with GDAL 2.3, 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). ‘different
 - {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, MULTIPOINT, 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*.

-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 *-nl*.

3.3.6.3 Examples

Create a VRT with a layer for each input shapefiles

```
ogrmerge.py -f VRT -o merged.vrt *.shp
```

Same, but creates a GeoPackage file

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

Concatenate the content of *france.shp* and *germany.shp* in *merged.shp*, and adds a ‘country’ field to each feature whose value is ‘france’ or ‘germany’ depending where it comes from.

```
ogrmerge.py -single -o merged.shp france.shp germany.shp -src_layer_field_name country
```

- *Common options*
- *ogrinfo*: Lists information about an OGR-supported data source.
- *ogr2ogr*: Converts simple features data between file formats.
- *ogrindex*: Creates a tileindex.
- *ogrlneref*: Create linear reference and provide some calculations using it.
- *ogrmerge*: Merge several vector datasets into a single one.

3.4 Geographic network programs

3.4.1 gnmmmanage

3.4.1.1 Synopsis

```
gnmmmanage [--help] [-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> ...]]
```

3.4.1.2 Description

The **gnmmmanage** 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.

-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 vertexes 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.

3.4.2 gnmanalyse

3.4.2.1 Synopsis

```
gnmanalyse [--help][--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>
```

3.4.2.2 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.

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)

- *gnmmanage*: Manages networks
- *gnmanalyse*: Analyses networks

RASTER DRIVERS

Short name	Long name	Creation	Copy	Geo-referencing	Build requirements
<i>AAIGrid</i>	Arc/Info ASCII Grid	No	Yes	Yes	Built-in by default
<i>ACE2</i>	ACE2	No	No	Yes	Built-in by default
<i>ADRG</i>	ADRG/ARC Digitized Raster Graphics (.gen/.thf)	Yes	Yes	Yes	Built-in by default
<i>AIG</i>	Arc/Info Binary Grid	No	No	Yes	Built-in by default
<i>AIRSAR</i>	AIRSAR Polarimetric Format	No	No	No	Built-in by default
<i>ARG</i>	Azavea Raster Grid	Yes	Yes	Yes	Built-in by default
<i>BAG</i>	Bathymetry Attributed Grid	No	Yes	Yes	libhdf5
<i>BLX</i>	Magellan BLX Topo File Format	No	Yes	Yes	Built-in by default
<i>BMP</i>	Microsoft Windows Device Independent Bitmap	Yes	Yes	Yes	Built-in by default
<i>BPG</i>	Better Portable Graphics	No	No	No	libbpg (manual build required for now)
<i>BSB</i>	Maptech/NOAA BSB Nautical Chart Format	No	No	Yes	Built-in by default
<i>BT</i>	VTP .bt Binary Terrain Format	Yes	Yes	Yes	Built-in by default
<i>BYN</i>	Natural Resources Canada's Geoid file format (.byn)	Yes	Yes	Yes	Built-in by default
<i>CAD</i>	AutoCAD DWG raster layer	No	No	Yes	(internal libopencad provided)
<i>CALS</i>	CALS Type 1	No	Yes	No	Built-in by default
<i>CEOS</i>	CEOS Image	No	Yes	No	Built-in by default
<i>COASP</i>	DRDC COASP SAR Processor Raster	No	No	No	Built-in by default
<i>COG</i>	Cloud Optimized GeoTIFF generator	No	Yes	Yes	Built-in by default
<i>COSAR</i>	TerraSAR-X Complex SAR Data Product	No	No	No	Built-in by default
<i>CPG</i>	Convair PolGASP data	No	No	Yes	Built-in by default
<i>CTable2</i>	CTable2 Datum Grid Shift	Yes	Yes	Yes	Built-in by default
<i>CTG</i>	USGS LULC Composite Theme Grid	No	No	Yes	Built-in by default
<i>DAAS</i>	DAAS (Airbus DS Intelligence Data As A Service driver)	No	No	Yes	libcurl
<i>DB2</i>	DB2 raster	Yes	Yes	Yes	ODBC (and any or all of PNG, JPEG, WEBP drivers)
<i>DDS</i>	DirectDraw Surface	No	Yes	No	Crunch Lib
<i>DERIVED</i>	Derived subdatasets driver	No	No	No	Built-in by default

Continued on next page

Table 1 – continued from previous page

Short name	Long name	Creation	Copy	Geo-referencing	Build requirements
<i>DIMAP</i>	Spot DIMAP	No	No	Yes	Built-in by default
<i>DIPEx</i>	ELAS DIPEx	No	No	Yes	Built-in by default
<i>DODS</i>	OPeNDAP Grid Client	No	No	Yes	libdap
<i>DOQ1</i>	First Generation USGS DOQ	No	No	Yes	Built-in by default
<i>DOQ2</i>	New Labelled USGS DOQ	No	No	Yes	Built-in by default
<i>DTED</i>	Military Elevation Data	No	Yes	Yes	Built-in by default
<i>E00GRID</i>	Arc/Info Export E00 GRID	No	No	Yes	Built-in by default
<i>ECRGTOC</i>	ECRG Table Of Contents (TOC.xml)	No	No	Yes	Built-in by default
<i>ECW</i>	Enhanced Compressed Wavelets (.ecw)	Yes	Yes	Yes	ECW SDK
<i>EEDAI</i>	Google Earth Engine Data API Image	No	No	Yes	libcurl
<i>EHdr</i>	ESRI .hdr Labelled	Yes	Yes	Yes	Built-in by default
<i>EIR</i>	Erdas Imagine Raw	No	No	Yes	Built-in by default
<i>ELAS</i>	Earth Resources Laboratory Applications Software	Yes	Yes	Yes	Built-in by default
<i>ENVI</i>	ENVI .hdr Labelled Raster	Yes	Yes	Yes	Built-in by default
<i>EPSILON</i>	Wavelet compressed images	No	Yes	No	epsilon 0.9.1
<i>ERS</i>	ERMapper .ERS	Yes	Yes	Yes	Built-in by default
<i>ESAT</i>	Envisat Image Product	No	No	No	Built-in by default
<i>EXR</i>	Extended Dynamic Range Image File Format	Yes	Yes	Yes	libopenexr
<i>FAST</i>	EOSAT FAST Format	No	No	Yes	Built-in by default
<i>FIT</i>	FIT	No	Yes	Yes	Built-in by default
<i>FITS</i>	Flexible Image Transport System	Yes	Yes	Yes	libfitsio
<i>FujiBAS</i>	Fuji BAS Scanner Image	No	No	No	Built-in by default
<i>GenBin</i>	Generic Binary (.hdr labelled)	No	No	No	Built-in by default
<i>GeoRaster</i>	Oracle Spatial GeoRaster	Yes	Yes	Yes	Oracle client libraries
<i>GFF</i>	Sandia National Laboratories GSAT File Format	No	No	No	Built-in by default
<i>GIF</i>	Graphics Interchange Format	No	Yes	No	(internal GIF library provided)
<i>GMT</i>	GMT Compatible netCDF	No	Yes	Yes	libnetcdf
<i>GPKG</i>	GeoPackage raster	Yes	Yes	Yes	libsqlite3 (and any or all of PNG, JPEG, WEBP drivers)
<i>GRASS</i>	GRASS Raster Format	No	No	Yes	libgrass
<i>GRASSASCII</i>	GRASS ASCII Grid	No	No	Yes	Built-in by default
<i>GRIB</i>	WMO General Regularly-distributed Information in Binary form	No	No	Yes	Built-in by default
<i>GS7BG</i>	Golden Software Surfer 7 Binary Grid File Format	Yes	Yes	Yes	Built-in by default
<i>GSAG</i>	Golden Software ASCII Grid File Format	No	No	Yes	Built-in by default
<i>GSBG</i>	Golden Software Binary Grid File Format	Yes	Yes	Yes	Built-in by default
<i>GSC</i>	GSC Geogrid	No	No	No	Built-in by default
<i>GTA</i>	Generic Tagged Arrays	No	Yes	Yes	libgta
<i>GTiff</i>	GeoTIFF File Format	Yes	Yes	Yes	Built-in by default
<i>GXF</i>	Grid eXchange File	No	No	Yes	Built-in by default

Continued on next page

Table 1 – continued from previous page

Short name	Long name	Creation	Copy	Geo-referencing	Build requirements
<i>HDF4</i>	Hierarchical Data Format Release 4 (HDF4)	Yes	Yes	Yes	libdf
<i>HDF5</i>	Hierarchical Data Format Release 5 (HDF5)	No	No	Yes	libhdf5
<i>HF2</i>	HF2/HFZ heightfield raster	No	Yes	Yes	Built-in by default
<i>HFA</i>	Erdas Imagine .img	Yes	Yes	Yes	Built-in by default
<i>IDA</i>	Image Display and Analysis	Yes	Yes	Yes	Built-in by default
<i>RST</i>	Idrisi Raster Format	Yes	Yes	Yes	Built-in by default
<i>IGNF</i>	IGN-France height correction ASCII grids	No	No	Yes	Built-in by default
<i>ILWIS</i>	Raster Map	Yes	Yes	Yes	Built-in by default
<i>INGR</i>	Intergraph Raster Format	Yes	Yes	Yes	Built-in by default
<i>IRIS</i>	Vaisala's weather radar software format	No	No	Yes	Built-in by default
<i>ISCE</i>	ISCE	Yes	Yes	Yes	Built-in by default
<i>ISG</i>	International Service for the Geoid	No	No	Yes	Built-in by default
<i>ISIS2</i>	USGS Astrogeology ISIS Cube (Version 2)	Yes	Yes	Yes	Built-in by default
<i>ISIS3</i>	USGS Astrogeology ISIS Cube (Version 3)	Yes	Yes	Yes	Built-in by default
<i>JDEM</i>	Japanese DEM (.mem)	No	No	Yes	Built-in by default
<i>JP2ECW</i>	ERDAS JPEG2000 (.jp2)	Yes	Yes	Yes	ECW SDK
<i>JP2KAK</i>	JPEG-2000 (based on Kakadu)	No	Yes	Yes	Kakadu library
<i>JP2LURA</i>	JPEG2000 driver based on Lurawave library	No	Yes	Yes	Lurawave library
<i>JP2MrSID</i>	JPEG2000 via MrSID SDK	No	Yes	Yes	MrSID SDK
<i>JP2OpenJPEG</i>	JPEG2000 driver based on OpenJPEG library	No	Yes	Yes	openjpeg >= 2.1
<i>JPEG</i>	JPEG JFIF File Format	No	Yes	Yes	(internal libjpeg provided)
<i>JPEG2000</i>	Implementation of the JPEG-2000 part 1	No	Yes	Yes	libjasper
<i>JPEGLS</i>	JPEGLS	No	Yes	No	CharLS library
<i>JPIPKAK</i>	JPIP Streaming	No	No	Yes	Kakadu library
<i>KEA</i>	KEA	Yes	Yes	Yes	libkea and libhdf5 libraries
<i>KMLSuperoverlay</i>	KML Superoverlay	No	Yes	Yes	Built-in by default
<i>KRO</i>	KOLOR Raw format	Yes	Yes	No	Built-in by default
<i>LIB</i>	NOAA Polar Orbiter Level 1b Data Set (AVHRR)	No	No	Yes	Built-in by default
<i>LAN</i>	Erdas 7.x .LAN and .GIS	No	No	Yes	Built-in by default
<i>LCP</i>	FARSITE v.4 LCP Format	No	Yes	Yes	Built-in by default
<i>Leveller</i>	Daylon Leveller Heightfield	Yes	Yes	Yes	Built-in by default
<i>LOSLAS</i>	NADCON .los/.las Datum Grid Shift	No	No	Yes	Built-in by default
<i>MAP</i>	OziExplorer .MAP	No	No	Yes	Built-in by default
<i>MRF</i>	Meta Raster Format	Yes	Yes	Yes	Built-in by default
<i>MBTiles</i>	MBTiles	Yes	Yes	Yes	libsqlite3
<i>MEM</i>	In Memory Raster	Yes	Yes	Yes	Built-in by default
<i>MFF</i>	Vexcel MFF Raster	Yes	Yes	Yes	Built-in by default
<i>MFF2</i>	Vexcel MFF2 Image	Yes	Yes	Yes	Built-in by default

Continued on next page

Table 1 – continued from previous page

Short name	Long name	Creation	Copy	Geo-referencing	Build requirements
<i>MG4Lidar</i>	MrSID/MG4 LiDAR Compression / Point Cloud View files	No	No	No	LIDAR SDK
<i>MrSID</i>	Multi-resolution Seamless Image Database	No	No	Yes	MrSID SDK
<i>MSG</i>	Meteosat Second Generation	No	No	Yes	msg library
<i>MSGN</i>	Meteosat Second Generation (MSG) Native Archive Format (.nat)	No	No	Yes	Built-in by default
<i>NDF</i>	NLAPS Data Format	No	No	Yes	Built-in by default
<i>netCDF</i>	NetCDF: Network Common Data Form	Yes	Yes	Yes	libnetcdf
<i>NGSGEOID</i>	NOAA NGS Geoid Height Grids	No	No	Yes	Built-in by default
<i>NGW</i>	NextGIS Web	No	No	Yes	libcurl
<i>NITF</i>	National Imagery Transmission Format	Yes	Yes	Yes	Built-in by default
<i>NTv1</i>	NTv1 Datum Grid Shift	No	No	Yes	Built-in by default
<i>NTv2</i>	NTv2 Datum Grid Shift	Yes	Yes	Yes	Built-in by default
<i>NWT_GRD</i>	Northwood/Vertical Mapper File Format	Yes	Yes	Yes	Built-in by default
<i>NWT_GRC</i>	Northwood/Vertical Mapper File Format	Yes	Yes	Yes	Built-in by default
<i>OZI</i>	OZF2/OZFX3 raster	No	No	Yes	Built-in by default
<i>JAXAPALSAR</i>	JAXA PALSAR Processed Products	No	No	Yes	Built-in by default
<i>PAux</i>	PCI .aux Labelled Raw Format	Yes	Yes	No	Built-in by default
<i>PCIDSK</i>	PCI Geomatics Database File	Yes	Yes	Yes	Built-in by default
<i>PCRaster</i>	PCRaster raster file format	Yes	Yes	Yes	(internal libcf provided)
<i>PDF</i>	Geospatial PDF	No	Yes	Yes	none for write support, Poppler/PoDoFo/PDFium for read support
<i>PDS</i>	Planetary Data System v3	No	No	Yes	Built-in by default
<i>PDS4</i>	NASA Planetary Data System (Version 4)	Yes	Yes	Yes	Built-in by default
<i>PLMosaic</i>	PLMosaic (Planet Labs Mosaics API)	No	No	Yes	libcurl
<i>PNG</i>	Portable Network Graphics	No	Yes	Yes	Built-in by default
<i>PNM</i>	Netpbm (.pgm, .ppm)	No	Yes	No	Built-in by default
<i>PostGISRaster</i>	PostGIS Raster driver	No	Yes	Yes	PostgreSQL library
<i>PRF</i>	PHOTOMOD Raster File	No	No	No	Built-in by default
<i>R</i>	R Object Data Store	No	Yes	No	Built-in by default
<i>Rasdaman</i>	Rasdaman GDAL driver	No	No	No	raslib
<i>Rasterlite</i>	Rasters in SQLite DB	No	Yes	Yes	libsqlite3
<i>SQLite</i>	Rasters in SQLite DB	No	Yes	Yes	libsqlite3, librasterlite2, libspatialite
<i>RDA</i>	RDA (DigitalGlobe Raster Data Access)	No	No	Yes	libcurl
<i>RDB</i>	RIEGL Database	No	No	Yes	rdlib >= 2.2.0.
<i>RIK</i>	Swedish Grid Maps	No	No	Yes	(internal zlib is used if necessary)
<i>RMF</i>	Raster Matrix Format	Yes	Yes	Yes	Built-in by default
<i>ROI_PAC</i>	ROI_PAC	Yes	Yes	Yes	Built-in by default
<i>RPFTOC</i>	Raster Product Format/RPF (a.toc)	No	No	Yes	Built-in by default

Continued on next page

Table 1 – continued from previous page

Short name	Long name	Creation	Copy	Geo-referencing	Build requirements
<i>RRASTER</i>	R Raster	Yes	Yes	Yes	Built-in by default
<i>RS2</i>	RadarSat 2 XML Product	No	No	Yes	Built-in by default
<i>SAFE</i>	Sentinel-1 SAFE XML Product	No	No	Yes	Built-in by default
<i>SAR_CEOS</i>	SARCEOS SAR Image	No	No	Yes	Built-in by default
<i>SAGA</i>	SAGA GIS Binary Grid File Format	Yes	Yes	Yes	Built-in by default
<i>SDE</i>	ESRI ArcSDE Raster	No	No	Yes	ESRI SDE
<i>SDTS</i>	USGS SDTS DEM	No	No	Yes	Built-in by default
<i>SENTINEL2</i>	Sentinel-2 Products	No	No	Yes	Built-in by default
<i>SGI</i>	SGI Image Format	Yes	Yes	No	Built-in by default
<i>SIGDEM</i>	Scaled Integer Gridded DEM	No	Yes	Yes	Built-in by default
<i>SNODAS</i>	Snow Data Assimilation System	No	No	Yes	Built-in by default
<i>SRP</i>	Standard Product Format (ASRP/USRP) (.gen)	No	No	Yes	Built-in by default
<i>SRTMHGT</i>	SRTM HGT Format	No	No	Yes	Built-in by default
<i>Terragen</i>	Terragen™ Terrain File	Yes	Yes	Yes	Built-in by default
<i>TIL</i>	EarthWatch/DigitalGlobe .TIL	No	No	Yes	Built-in by default
<i>TileDB</i>	TileDB	Yes	Yes	Yes	TileDB
<i>TSX</i>	TerraSAR-X Product	No	No	Yes	Built-in by default
<i>USGSDEM</i>	USGS ASCII DEM (and CDED)	Yes	No	Yes	Built-in by default
<i>VICAR</i>	VICAR	Yes	Yes	Yes	Built-in by default
<i>VRT</i>	GDAL Virtual Format	Yes	Yes	Yes	Built-in by default
<i>WCS</i>	OGC Web Coverage Service	No	No	Yes	libcurl
<i>WEBP</i>	WEBP	No	Yes	No	libwebp
<i>WMS</i>	Web Map Services	No	Yes	Yes	libcurl
<i>WMTS</i>	OGC Web Map Tile Service	No	No	Yes	libcurl
<i>XPM</i>	X11 Pixmap	No	Yes	No	Built-in by default
<i>XYZ</i>	ASCII Gridded XYZ	No	Yes	Yes	Built-in by default
<i>ZMAP</i>	ZMap Plus Grid	No	Yes	Yes	Built-in by default

4.1 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.

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. From GDAL 1.8.0, you can explicitly specify the datatype by setting the AAIGRID_DATATYPE configuration option (Int32, Float32 and Float64 values are supported currently)

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. To 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.

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 behaviour 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). Starting with GDAL 1.11, 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 `gdal/frmts/aaigrid/aaigriddataset.cpp`.

4.1.1 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.*)

4.2 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 `gdal/frmts/raw/ace2dataset.cpp`.

4.2.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.3 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. Creation is possible, but it must be considered as experimental and a means of testing read access (although files created by the driver can be read successfully on another GIS software)

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 \(MIL-A-89007\)](#)

4.3.1 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.)*

4.4 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 `gdal/frmts/aigrid/aigdataset.cpp`.

4.4.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.4.1.1 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 FFFFC14 00000000 00000000 ~ ~ ! ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
16: 00000000 00000000 0000BFA 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 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
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 FFFFC14 00000000 00000000 ~ ~ ' ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
16: 00000000 00000000 00016DAE 00000000 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ m ~ ~ ~ ~ ~
32: 00000000 00000000 00000000 00000000 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
48: 00000000 00000000 00000000 00000000 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
64: 00000000 00000000 00000000 00000000 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
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 $65536 - \text{byte0} * 256 - \text{byte1}$ if $\text{byte0} > 127$.

$104 + \text{RMinSize}$, ...

$\text{RTileSize} * 2 - 3 - \text{RMinSize}$

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
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

HFilesPerRow

The width of the file in tiles (often 8 for files of less than 2K in width).

292

4

MSB Int32

HFilesPerColumn

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.
- Nouredine 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.

4.5 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) = $HH * \text{conj}(HH)$
- Band 2: Covariance_12 (CFloat32) = $\sqrt{2} * HH * \text{conj}(HV)$
- Band 3: Covariance_13 (CFloat32) = $HH * \text{conj}(VV)$
- Band 4: Covariance_22 (Float32) = $2 * HV * \text{conj}(HV)$
- Band 5: Covariance_23 (CFloat32) = $\sqrt{2} * HV * \text{conj}(VV)$
- Band 6: Covariance_33 (Float32) = $VV * \text{conj}(VV)$

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_1.mocomp*, *mission_meta.airsar* or *mission_meta.podaac*.

4.5.1 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations (vsimem/, etc.)*

4.5.2 See Also

- [AIRSAR Data Format](#)

4.6 ARG – Azavea Raster Grid

Driver short name

ARG

Driver built-in by default

This driver is built-in by default

Driver implementation for a raw format that is used in [GeoTrellis](#) and called ARG. [ARG format specification](#). Format is essentially a raw format, with a companion .JSON file.

NOTE: Implemented as `gdal/frmts/arg/argdataset.cpp`.

4.6.1 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.)*

4.7 BAG – Bathymetry Attributed Grid

Driver short name

BAG

Build dependencies

libhdf5

This driver provides read-only support, and starting with GDAL 2.4 for creation, 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 or three image bands representing Elevation (band 1), Uncertainty (band 2) and Nominal Elevation (band 3) values for each cell in a raster grid area.

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.

4.7.1 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.)*

4.7.2 Variable resolution (VR) grid support

Starting with GDAL 2.4, 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**: this is the default mode. 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 of 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. Interpolation of cells at nodata value can also be done using a inverse distance weighting interpolation. 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 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`: 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 is 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 use the mean value of the elevation of intersecting nodes, and the maximum uncertainty of those nodes.
- `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.
- `INTERPOLATION=NO/INVDIST`. Default to `NO`. If set to `INVDIST`, a inverse distance weighting interpolation of nodata values is applied after the above describe value population. Interpolation cannot be used together with `SUPERGRIDS_MASK=YES`.
- `NODATA_VALUE=value`. Override the default value, which is usually 1000000.

4.7.3 Creation support

Starting with GDAL 2.4, 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. Currently those creation options are:

- `VAR_INDIVIDUAL_NAME=string`: to fill `contact/CI_ResponsibleParty/individualName`. If not provided, default to “unknown”.
- `VAR_ORGANISATION_NAME=string`: to fill `contact/CI_ResponsibleParty/organisationName`. If not provided, default to “unknown”.

- `VAR_POSITION_NAME=string`: to fill `contact/CI_ResponsibleParty/positionName`. If not provided, default to “unknown”.
- `VAR_DATE=YYYY-MM-DD`: to fill `dateStamp/Date`. If not provided, default to current date.
- `VAR_VERT_WKT=wkt_string`: to fill `referenceSystemInfo/MD_ReferenceSystem/referenceSystemIdentifier/RS_Identifier/code` 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=string`: to fill `identificationInfo/abstract`. If not provided, default to empty string
- `VAR_PROCESS_STEP_DESCRIPTION=string`: to fill `dataQualityInfo/lineage/LI_Lineage/processStep/LI_ProcessStep/description`. If not provided, default to “Generated by GDAL x.y.z”.
- `VAR_DATETIME=YYYY-MM-DDTHH:MM:SS` : to fill `dataQualityInfo/lineage/LI_Lineage/processStep/LI_ProcessStep/dateT`. If not provided, default to current datetime.
- `VAR_RESTRICTION_CODE=enumerated_value`: to fill `metadataConstraints/MD_LegalConstraints/useConstraints/MD_RestrictionCode`. If not provided, default to “otherRestrictions”.
- `VAR_OTHER_CONSTRAINTS=string`: to fill `metadataConstraints/MD_LegalConstraints/otherConstraints`. If not provided, default to “unknown”.
- `VAR_CLASSIFICATION=enumerated_value`: to fill `metadataConstraints/MD_SecurityConstraints/classification/MD_ClassificationCode`. If not provided, default to “unclassified”.
- `VAR_SECURITY_USER_NOTE=string`: to fill `metadataConstraints/MD_SecurityConstraints/userNote`. If not provided, default to “none”.

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 substituable 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=string`: Value to write in the `/BAG_root/BAG Version` attribute. Default to 1.6.2.
- `COMPRESS=NONE/DEFLATE`: Compression for elevation and uncertainty grids. Default to DEFLATE.
- `ZLEVEL=[1-9]`: Deflate compression level. Defaults to 6.
- `BLOCK_SIZE=value_in_pixel`: Chunking size of the HDF5 arrays. Default to 100, or the maximum dimension of the raster if smaller than 100.

4.7.4 Usage examples

- Opening in low resolution mode:

```
$ gdalinfo data/test_vr.bag

[...]  
Size is 6, 4  
[...]  
  HAS_SUPERGRIDS=TRUE  
  MAX_RESOLUTION_X=29.900000  
  MAX_RESOLUTION_Y=31.900000  
  MIN_RESOLUTION_X=4.983333
```

(continues on next page)

(continued from previous page)

```
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, no interpolation):

```
$ 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, with interpolation of nodata values.

```
$ gdal_translate data/test_vr.bag -oo MODE=RESAMPLED_GRID -oo
  ↳INTERPOLATION=INVDIST 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 substituable 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"
```

4.7.5 See Also

- Implemented as `gdal/frmts/hdf5/bagdataset.cpp`.
- [The Open Navigation Surface Project](#)
- [Description of Bathymetric Attributed Grid Object \(BAG\) Version 1.6](#)
- [Variable resolution grid extension for BAG files](#)

4.8 BLX – Magellan BLX Topo File Format

Driver short name

BLX

Driver built-in by default

This driver is built-in by default

BLX is the format for storing topographic data in Magellan GPS units. This driver supports both reading and writing. In addition the 4 overview levels inherent in the BLX format can be used with the driver.

The BLX format is tile based, for the moment the tile size is fixed to 128x128 size. Furthermore the dimensions must be a multiple of the tile size.

The data type is fixed to Int16 and the value for undefined values is fixed to -32768. In the BLX format undefined values are only really supported on tile level. For undefined pixels in non-empty tiles see the `FILLUNDEF/FILLUNDEFVAL` options.

4.8.1 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.*)

4.8.2 Georeferencing

The BLX projection is fixed to WGS84 and georeferencing from BLX is supported in the form of one tiepoint and pixelsize.

4.8.3 Creation Issues

Creation Options:

- **ZSCALE=1**: Set the desired quantization increment for write access. A higher value will result in better compression and lower vertical resolution.
- **BIGENDIAN=YES**: If BIGENDIAN is defined, the output file will be in XLB format (big endian blx).
- **FILLUNDEF=YES**: If FILLUNDEF is yes the value of FILLUNDEFVAL will be used instead of -32768 for non-empty tiles. This is needed since the BLX format only support undefined values for full tiles, not individual pixels.
- **FILLUNDEFVAL=0**: See FILLUNDEF

4.9 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.

4.9.1 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.*)

4.9.2 Creation Options

- **WORLDFILE=YES:** Force the generation of an associated ESRI world file (with the extension `.wld`).

4.9.3 See Also

- Implemented as `gdal/frmts/bmp/bmpdataset.cpp`.
- [Wikipedia BMP file format](#)

4.10 BPG – Better Portable Graphics

Driver short name

BPG

Build dependencies

libbpg (manual build required for now)

NOTE: Implemented as `gdal/frmts/bpg/bpgdataset.cpp`.

4.11 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 frmts/bsb/bsb_read.c files for details on patents affecting BSB format.

Starting with GDAL 1.6.0, 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 `gdal/frmts/bsb/bsbdataset.cpp`.

4.11.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (vsimem/, etc.)*

4.11.2 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 outline of the chart.

4.12 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 `gdal/frmts/raw/btdataset.cpp`.

See Also: The [BT file format](#) is defined on the [VTP](#) web site.

4.12.1 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.*)

4.13 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.

4.13.1 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.*)

4.13.2 Factor

When translating from or into 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 the scale 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**. ($\text{Scale} = 1.0 / \text{Factor}$).

4.13.3 See Also

- Implemented as `gdal/frmts/raw/byndataset.{h,cpp}`.
- www.nrcan.gc.ca

4.14 CAD – AutoCAD DWG raster layer

Driver short name

CAD

New in version 2.2.

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 metadata from DWG Image description and set it to GDALDataset.

NOTE: Implemented as `ogr/ogrsf_frmts/cad/gdalcaddataset.cpp`.

4.14.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

4.15 CALS – CALS Type 1

Driver short name

CALS

New in version 2.1.

Driver built-in by default

This driver is built-in by default

CALS Type 1 rasters are untiled back 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).

4.15.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

4.15.2 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.

4.15.3 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.

4.15.4 See Also

- [MIL-PRF-28002C](#)
- [MIL-STD-1840C](#)

4.16 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* driver for access to SAR CEOS data products.

NOTE: Implemented as `gdal/frmts/ceos/ceosdataset.cpp`.

4.16.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.17 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 `gdal/frmts/coasp/coasp_dataset.cpp`.

4.18 COG – Cloud Optimized GeoTIFF generator

New 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)

4.18.1 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.*)

4.18.2 Creation Options

4.18.2.1 General creation options

- **BLOCKSIZE=n**: Sets the tile width and height in pixels. Defaults to 512.
- **COMPRESS=[NONE/LZW/JPEG/DEFLATE/ZSTD/WEBP/LERC/LERC_DEFLATE/LERC_ZSTD]**: Set the compression to use.
 - 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). See the “[8 and 12 bit JPEG in TIFF](#)” wiki page for more details. 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.
 - 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 is available when using internal libtiff.
 - LERC_ZSTD is available when LERC and ZSTD are available.
- **LEVEL=integer_value**: DEFLATE/ZSTD/LERC_DEFLATE/LERC_ZSTD compression level. A lower number will result in faster compression but less efficient compression rate. 1 is the fastest. For DEFLATE, 9 is the slowest/higher compression rate (the default is 6) For ZSTD, 22 is the slowest/higher compression rate (the default is 9)
- **MAX_Z_ERROR=threshold**: Set the maximum error threshold on values for LERC/LERC_DEFLATE/LERC_ZSTD compression. The default is 0 (lossless).
- **QUALITY=integer_value**: JPEG/WEBP quality setting. A value of 100 is best quality (least compression), and 1 is worst quality (best compression). The default is 75. For WEBP, QUALITY=100 automatically turns on lossless mode.
- **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.

Note: Overview generation by itself, which can take most of the total processing time, is not multithreaded currently.

- **PREDICTOR=[YES/NO/STANDARD/FLOATING_POINT]:** Set the predictor for LZW, DEFLATE and ZSTD compression. The default is NO. 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 know 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]:** Resampling method used for overview generation or reprojection. For paletted images, NEAREST is used by default, otherwise it is CUBIC.
- **OVERVIEWS=[AUTO/IGNORE_EXISTING/FORCE_USE_EXISTING/NONE]:** Describe the behaviour regarding overview generation and use of source overviews.
 - AUTO (default): source overviews will be used if present (even if the dimension of the smallest level is not < 512 pixels), and, 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 (even if the dimension of the smallest level is not < 512 pixels). 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.

4.18.2.2 Reprojection related creation options

- **TILING_SCHEME=CUSTOM/GoogleMapsCompatible:** If set to `GoogleMapsCompatible`, reprojection to EPSG:3857 using a `GoogleMapsCompatible` tiling scheme will be automatically done. The default block size in that case will be 256. If explicitly setting another block size, this one will be taken into account (that is if setting a higher value than 256, the original `GoogleMapsCompatible` tiling scheme is modified to take into account the size of the HiDiPi tiles). In `GoogleMapsCompatible` mode, `TARGET_SRS`, `RES` and `EXTENT` options are ignored.
- **TARGET_SRS=string:** 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=INT:** Number of overview levels for which GeoTIFF tile and WebMercator tiles 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 `GoogleMapsCompatible` tiling scheme. Only taken into account if `TILING_SCHEME=GoogleMapsCompatible`. As up to $2^{\text{ALIGNED_LEVELS}}$ tiles can be added in each dimension, it is the responsibility of the user to use this setting with care (a hard limit of 10 is enforced by the driver).
- **ADD_ALPHA=YES/NO:** Whether an alpha band is added in case of reprojection. Defaults to YES.
- **GEOTIFF_VERSION=[AUTO/1.0/1.1]:** (GDAL >= 3.1.0) 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.

4.18.3 File format details

4.18.3.1 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.

4.18.3.2 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)

4.18.3.3 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 `TileOffset[i]` and of size `TileByteCount[i]`)
 - trailer of imagery (4 bytes)
 - leader with mask tilesize (4 bytes)
 - mask data (starting at `mask.TileOffset[i]` and of size `mask.TileByteCount[i]`, but none of them actually need to be read)
 - trailer of mask data (4 bytes)

4.18.3.4 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`.

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`)

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 `TileOffset[i] + TileByteCount[i]`.

4.18.4 Examples

```
gdalwarp src1.tif src2.tif out.tif -of COG
```

```
gdal_translate world.tif world_webmerc_cog.tif -of COG -co TILING_
↪SCHEME=GoogleMapsCompatible -co COMPRESS=JPEG
```

4.18.5 See Also

- *GTiff – GeoTIFF File Format* driver
- How to generate and read cloud optimized GeoTIFF files (before GDAL 3.1)

4.19 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.

4.19.1 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

4.19.2 See Also

- DLR Document TX-GS-DD-3307 “Level 1b Product Format Specification.”

4.20 CPG – Convair PolGASP data

Driver short name

CPG

Driver built-in by default

This driver is built-in by default

NOTE: Implemented as `gdal/frmts/raw/cpgdataset.cpp`.

4.20.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (vsimem/, etc.)*

4.21 CTable2 – CTable2 Datum Grid Shift

Driver short name

CTable2

Driver built-in by default

This driver is built-in by default

NOTE: Implemented as `gdal/frmts/raw/ctable2dataset.cpp`.

4.21.1 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.)*

4.22 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 `gdal/frmts/ctg/ctgdataset.cpp`.

4.22.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.23 DAAS (Airbus DS Intelligence Data As A Service driver)

Driver short name

DAAS

New 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/>

4.23.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

4.23.2 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.

4.23.3 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` open 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` open 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

4.23.4 Open options

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**=value. Size of a block in pixels requested to the server. Defaults to 512 pixels. Between 64 and 8192.
- **PIXEL_ENCODING**=value. Format in which pixels are queried. Defaults to
 - **AUTO**: for 1, 3 or 4-band images of type Byte, resolves to PNG. Otherwise to RAW
 - **RAW**: compatible of all images. Pixels are requested in a uncompressed raw format.
 - **PNG**: compatible of 1, 3 or 4-band images of type Byte

- **JPEG**: compatible of 1 or 3-band images of type Byte
- **JPEG2000**: compatible of all images. Requires GDAL to be built with one of its JPEG2000-capable drivers.
- **MASKS=YES/NO**. Whether to expose mask bands. Default to YES.

4.24 DB2 raster

Driver short name

DB2

Build dependencies

ODBC (and any or all of PNG, JPEG, WEBP drivers)

Note: The DB2 driver is based largely on the source code for the OGR/DB2 GeoPackage (GPKG) driver, replacing SQLITE functionality with corresponding DB2 functionality. Appreciation is expressed for the major development required to produce the GeoPackage driver which dramatically reduced the effort to produce the DB2 driver. At some point it may be worthwhile to explore whether refactoring of the class structure to share common functionality is practical.

The DB2 driver largely implements the GeoPackage standard, with the main difference that the “gpkg_” prefix is replaced with “gpkg.” in the DB2 table name, assigning them to a distinct database schema.

The documentation below is largely copied from the *GPKG – GeoPackage raster* documentation. References to the GeoPackage standard have been left as such. References to the implementation have been changed to DB2. In some cases it isn’t clear whether we should refer to “DB2 tiles” or “GeoPackage tiles”.

Starting with GDAL 2.0, 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 DB2 tables in the DB2 database, so it must be run by a user with create authority on the database it is working with.

The driver can also handle DB2 vectors. See [DB2 vector](#) documentation page

Various kind of input datasets can be converted to DB2 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

DB2 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.

4.24.1 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

4.24.2 Opening options

By default, the driver will expose a DB2 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.

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.

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 DB2 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=1/2/3/4**: 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. Defaults to 4 (which is the always safe value).
- **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: Whether to use the extent of actual existing tiles at the zoom level of the full resolution dataset. Defaults to NO.
- **TILE_FORMAT**=PNG_JPEG/PNG/PNG8/JPEG/WEBP: Format used to store tiles. See *Tile format* section. Only used in update mode. Defaults to PNG_JPEG.
- **QUALITY**=1-100: Quality setting for JPEG and WEBP compression. Only used in update mode. Default to 75.
- **ZLEVEL**=1-9: DEFLATE compression level for PNG tiles. Only used in update mode. Default to 6.
- **DITHER**=YES/NO: Whether to use Floyd-Steinberg dithering (for **TILE_FORMAT**=PNG8). Only used in update mode. Defaults to NO.

Note: open options are typically specified with “-oo name=value” syntax in most GDAL utilities, or with the `GDALOpenEx()` API call.

4.24.3 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 DB2 dataset (seen as GDAL subdatasets), or to add a tile table to an existing vector-only DB2, 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 DB2 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 DB2 tile. In which case, each GDAL block will overlap four DB2 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 DB2 table to store partial DB2 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 DB2 table with the appropriate compression. All of this is transparent to the user of GDAL API/utilities

4.24.3.1 Tile formats

DB2 can store tiles in different formats, PNG and/or JPEG for the baseline specification, and WebP for extended DB2. 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 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.

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 ($\geq 0.1.4$) 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 that 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.

4.24.3.2 Tiling schemes

By default, conversion to DB2 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 DB2 with tile servicing software, it is possible to select a predefined tiling scheme that has world coverage. The available tiling schemes are :

- *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.
- *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 DB2 (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 in easting and northing in the range [-20037508.34,20037508.34]. The same remark as with *PseudoTMS_GlobalGeodetic* applies regarding interoperability with TMS.

In all the above tiling schemes, consecutive zoom levels defer by a resolution of a factor of two.

4.24.3.3 Creation options

The following creation options are available:

- **RASTER_TABLE**=string. Name of tile user table. By default, based on the source filename.
- **APPEND_SUBDATASET**=YES/NO: If set to YES, an existing DB2 table will not be priorly destroyed, such as to be able to add new content to it. Defaults to NO.
- **RASTER_IDENTIFIER**=string. Human-readable identifier (e.g. short name), put in the *identifier* column of the *gpkg.contents* table.
- **RASTER_DESCRIPTION**=string. Human-readable description, put in the *description* column of the *gpkg.contents* table.
- **BLOCKSIZE**=integer. Block size in width and height in pixels. Defaults to 256. Maximum supported is 4096. Should not be set when using a non-custom **TILING_SCHEME**.
- **BLOCKXSIZE**=integer. Block width in pixels. Defaults to 256. Maximum supported is 4096.
- **BLOCKYSIZE**=integer. Block height in pixels. Defaults to 256. Maximum supported is 4096.
- **TILE_FORMAT**=PNG_JPEG/PNG/PNG8/JPEG/WEBP: Format used to store tiles. See *Tile formats* section. Defaults to PNG_JPEG.
- **QUALITY**=1-100: Quality setting for JPEG and WEBP compression. Default to 75.
- **ZLEVEL**=1-9: DEFLATE compression level for PNG tiles. Default to 6.
- **DITHER**=YES/NO: Whether to use Floyd-Steinberg dithering (for **TILE_FORMAT**=PNG8). Defaults to NO.
- **TILING_SCHEME**=CUSTOM/GoogleCRS84Quad/GoogleMapsCompatible/InspireCRS84Quad/PseudoTMS_GlobalGeodetic: See *Tiling schemes* section. Defaults to CUSTOM.
- **ZOOM_LEVEL_STRATEGY**=AUTO/LOWER/UPPER. Strategy to determine zoom level. Only used for **TILING_SCHEME** is 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. Resampling algorithm. Only used for **TILING_SCHEME** is different from CUSTOM. Defaults to BILINEAR.

4.24.4 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`)

4.24.5 Metadata

GDAL uses the standardized ``gpkg.metadata`` [`<http://www.geopackage.org/spec/#_metadata_table>`](http://www.geopackage.org/spec/#_metadata_table) and ``gpkg.metadata_reference`` [`<http://www.geopackage.org/spec/#_metadata_reference_table>`](http://www.geopackage.org/spec/#_metadata_reference_table) 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 DB2, 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.

4.24.6 Examples

- Simple translation of a GeoTIFF into DB2. The table 'byte' will be created with the tiles.

```
% gdal_translate -of DB2ODBC byte.tif DB2ODBC:database=sample;DSN=SAMPLE
```

- Translation of a GeoTIFF into DB2 using WebP tiles

```
% gdal_translate -of DB2ODBC byte.tif DB2ODBC:database=sample;DSN=SAMPLE -co TILE_
↳FORMAT=WEBP
```

- Translation of a GeoTIFF into DB2 using GoogleMapsCompatible tiling scheme (with reprojection and resampling if needed)

```
% gdal_translate -of DB2ODBC byte.tif DB2ODBC:database=sample;DSN=SAMPLE -co_
↳TILING_SCHEME=GoogleMapsCompatible
```

- Building of overviews of an existing DB2

```
% gdaladdo -oo RASTER_TABLE=world -r cubic DB2ODBC:database=sample;DSN=SAMPLE 2 4_
↳8 16 32 64
```

- Addition of a new subdataset to an existing DB2, and choose a non default name for the raster table.

```
% gdal_translate -of DB2ODBC new.tif DB2ODBC:database=sample;DSN=SAMPLE -co_
↳APPEND_SUBDATASET=YES -co RASTER_TABLE=new_table
```

- Reprojection of an input dataset to DB2

```
% gdalwarp -of DB2ODBC -co RASTER_TABLE=new_table in.tif DB2ODBC:database=sample;
↳DSN=SAMPLE -t_srs EPSG:3857
```

- Open a specific raster table in a DB2

```
% gdalinfo DB2ODBC:database=sample;DSN=SAMPLE -oo TABLE=a_table
```

4.24.7 See Also

- [DB2 vector](#) documentation page
- [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

4.25 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](#)

NOTE: Implemented as `gdal/frmts/dds/ddsdataset.cpp`.

4.25.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

4.25.2 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

4.25.2.1 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
```

4.25.2.2 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
```

4.26 DERIVED – Derived subdatasets driver

Driver short name

DERIVED

Driver built-in by default

This driver is built-in by default

This driver allows to access to 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.

4.26.1 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.

4.26.2 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],
  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)

```

(continues on next page)

(continued from previous page)

```

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

```

4.26.3 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",
  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"]]

```

(continues on next page)

(continued from previous page)

```

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:
  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

```

4.26.4 See Also:

- *Using Derived Bands part of the GDAL VRT tutorial*

4.27 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 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.

From GDAL 1.6.0, 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.

NOTE: Implemented as `gdal/frmts/dimap/dimapdataset.cpp`.

4.27.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (vsimem/, etc.)*

4.28 DIPEX – ELAS DIPEX

Driver short name

DIPEX

Driver built-in by default

GDAL

This driver is built-in by default

NOTE: Implemented as `gdal/frmts/raw/dipxdataset.cpp`.

4.28.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.29 DODS – OPeNDAP Grid Client

Driver short name

DODS

Build dependencies

libdap

GDAL optionally includes read support for 2D grids and arrays via the OPeNDAP (DODS) protocol.

4.29.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

4.29.2 Dataset Naming

The full dataset name specification consists of the OPeNDAP dataset url, the full path to the desired array or grid variable, and an indicator of the array indices to be accessed.

For instance, if the url <http://maps.gdal.org/daac-bin/nph-hdf/3B42.HDF.dds> returns a DDS definition like this:

```
Dataset {  
  Structure {  
    Structure {  
      Float64 precipitate[scan = 5][longitude = 360][latitude = 80];  
      Float64 relError[scan = 5][longitude = 360][latitude = 80];  
    }  
  }  
}
```

(continues on next page)

(continued from previous page)

```

    } PlanetaryGrid;
  } DATA_GRANULE;
} 3B42.HDF;

```

then the precipitate grid can be accessed using the following GDAL dataset name:

http://maps.gdal.org/daac-bin/nph-hdf/3B42.HDF?DATA_GRANULE.PlanetaryGrid.precipitate{{{0}}}{{{x}}}{{{y}}}

The full path to the grid or array to be accessed needs to be specified (not counting the outer Dataset name). GDAL needs to know which indices of the array to treat as x (longitude or easting) and y (latitude or northing). Any other dimensions need to be restricted to a single value.

In cases of data servers with only 2D arrays and grids as immediate children of the Dataset it may not be necessary to name the grid or array variable.

In cases where there are a number of 2D arrays or grids at the dataset level, they may be each automatically treated as separate bands.

4.29.3 Specialized AIS/DAS Metadata

A variety of information will be transported via the DAS describing the dataset. Some DODS drivers (such as the GDAL based one!) already return the following DAS information, but in other cases it can be supplied locally using the AIS mechanism. See the DODS documentation for details of how the AIS mechanism works.

```

Attributes {

  GLOBAL {
    Float64 Northernmost_Northing 71.1722;
    Float64 Southernmost_Northing 4.8278;
    Float64 Easternmost_Easting -27.8897;
    Float64 Westernmost_Easting -112.11;
    Float64 GeoTransform "71.1722 0.001 0.0 -112.11 0.0 -0.001";
    String spatial_ref "GEOGCS[\"WGS 84\", DATUM[\"WGS_1984\", SPHEROID[\"WGS 84\",
↪6378137,298.257223563]], PRIMEM[\"Greenwich\",0], UNIT[\"degree\",0.0174532925199433]]
↪";
    Metadata {
      String TIFFTAG_XRESOLUTION "400";
      String TIFFTAG_YRESOLUTION "400";
      String TIFFTAG_RESOLUTIONUNIT "2 (pixels/inch)";
    }
  }

  band_1 {
    String Description "...";
    String
  }
}

```

4.29.3.1 Dataset

There will be an object in the DAS named GLOBAL containing attributes of the dataset as a whole.

It will have the following subitems:

- **Northernmost_Northing:** The latitude or northing of the north edge of the image.
- **Southernmost_Northing:** The latitude or northing of the south edge of the image.
- **Easternmost_Easting:** The longitude or easting of the east edge of the image.
- **Westernmost_Easting:** The longitude or easting of the west edge of the image.
- **GeoTransform:** The six parameters defining the affine transformation between pixel/line space and georeferenced space if applicable. Stored as a single string with values separated by spaces. Note this allows for rotated or sheared images. (optional)
- **SpatialRef:** The OpenGIS WKT description of the coordinate system. If not provided it will be assumed that the coordinate system is WGS84. (optional)
- **Metadata:** a container with a list of string attributes for each available metadata item. The metadata item keyword name will be used as the attribute name. Metadata values will always be strings. (optional)
- *address GCPs*

Note that the edge northing and easting values can be computed based on the grid size and the geotransform. They are included primarily as extra documentation that is easier to interpret by a user than the GeoTransform. They will also be used to compute a GeoTransform internally if one is not provided, but if both are provided the GeoTransform will take precedence.

4.29.3.2 Band

There will be an object in the DAS named after each band containing attribute of the specific band.

It will have the following subitems:

- **Metadata:** a container with a list of string attributes for each available metadata item. The metadata item keyword name will be used as the attribute name. Metadata values will always be strings. (optional)
- **PhotometricInterpretation:** Will have a string value that is one of “Undefined”, “GrayIndex”, “PaletteIndex”, “Red”, “Green”, “Blue”, “Alpha”, “Hue”, “Saturation”, “Lightness”, “Cyan”, “Magenta”, “Yellow” or “Black”. (optional)
- **units:** name of units (one of “ft” or “m” for elevation data). (optional)
- **add_offset:** Offset to be applied to pixel values (after scale_factor) to compute a “real” pixel value. Defaults to 0.0. (optional)
- **scale_factor:** Scale to be applied to pixel values (before add_offset) to compute “real” pixel value. Defaults to 1.0. (optional)
- **Description:** Descriptive text about the band. (optional)
- **missing_value:** The nodata value for the raster. (optional)
- **Colormap:** A container with a subcontainer for each color in the color table, looking like the following. The alpha component is optional and assumed to be 255 (opaque) if not provided.

```
Colormap {
  Color_0 {
    Byte red 0;
```

(continues on next page)

(continued from previous page)

```

    Byte green 0;
    Byte blue 0;
    Byte alpha 255;
}
Color_1 {
    Byte red 255;
    Byte green 255;
    Byte blue 255;
    Byte alpha 255;
}
...
}

```

4.29.4 See Also

- [OPeNDAP Website](#)

4.30 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 `gdal/frmts/raw/doq1dataset.cpp`.

4.30.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.31 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 `gdal/frmts/raw/doq2dataset.cpp`.

See Also: [USGS DOQ Standards](#)

4.31.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.32 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.

4.32.1 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.*)

4.32.2 Read Issues

4.32.2.1 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.

4.32.2.2 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.

4.32.2.3 Configuration options

This paragraph lists the configuration options that can be set to alter the default behaviour of the DTED driver.

- `REPORT_COMPD_CS`: (GDAL \geq 2.2.2). Can be set to `TRUE` to avoid stripping the vertical CS of compound CS when reading the SRS of a file. Default value : `FALSE`

4.32.2.4 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.

4.32.3 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.

4.32.4 GeoTransform

The `DTED_APPLY_PIXEL_IS_POINT=TRUE` environment variable can be set (GDAL \geq 3.1) to apply a pixel-is-point interpretation to the data when reading the geotransform.

4.32.5 See Also

- Implemented as `gdal/frmts/dted/dteddataset.cpp`.

4.33 E00GRID – Arc/Info Export E00 GRID

Driver short name

E00GRID

Driver built-in by default

This driver is built-in by default

GDAL supports reading DEMs/rasters exported as E00 Grids.

The driver has been tested against datasets such as the one available on <ftp://msdis.missouri.edu/pub/dem/24k/county/>

NOTE: Implemented as `gdal/frmts/e00grid/e00griddataset.cpp`.

4.33.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (vsimem/, etc.)*

4.34 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
```

4.34.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (vsimem/, etc.)*

4.34.2 See Also

- *NITF – National Imagery Transmission Format*: format of the ECRG frames
- [MIL-PRF-32283](#) : specification of ECRG products

NOTE: Implemented as `gdal/frmts/nitf/ecrgtodata.cpp`

4.35 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](#).

4.35.1 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.*)

4.35.2 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.

4.35.3 History

- v3.x - Last release, 2006
- v4.x - Last release, 2012
- v5.x - Active development, 2013 - current

4.35.4 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”.

4.35.5 Creation Options:

- **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:** 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=name:** Name of the ECW projection string to use. Common examples are NUTM11, or GEODETIC.
- **DATUM=name:** Name of the ECW datum string to use. Common examples are WGS84 or NAD83.
- **UNITS=name:** (GDAL >= 1.9.0) Name of the ECW projection units to use : METERS (default) or FEET (us-foot).

- **ECW_FORMAT_VERSION=2/3**: (GDAL >= 1.10.0) When build 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.

4.35.6 Configuration Options

The ERDAS ECW SDK supports a variety of [runtime configuration options](#) to control various features. Most of these are exposed as GDAL configuration options. 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=number_of_megabytes**: The maximum number of megabytes of space in the ECWP_CACHE_LOCATION to be used for caching ECWP results.
- **ECWP_BLOCKING_TIME_MS**: time an ecwp:// blocking read will wait before returning - default 10000 ms.
- **ECWP_REFRESH_TIME_MS**: time delay between blocks arriving and the next refresh callback - default 10000 ms. 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**: This may be set to FALSE to disable dithering when decompressing ECW files. Defaults to TRUE.
- **ECW_FORCE_FILE_REOPEN=TRUE/FALSE**: This may be set to TRUE to force open a file handle for each file for each connection made. Defaults to FALSE.
- **ECW_CACHE_MAXOPEN=number**: The maximum number of files to keep open for ECW file handle caching. Defaults to unlimited.
- **ECW_RESILIENT_DECODING=TRUE/FALSE**: Controls whether the reader should be forgiving of errors in a file, trying to return as much data as is available. Defaults to TRUE. If set to FALSE an invalid file will result in an error.

The GDAL-specific options:

- **ECW_ALWAYS_UPWARD=TRUE/FALSE**: 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.

4.35.6.1 ECW Version 3 Files

(Starting with GDAL 1.10.0)

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.

4.35.6.2 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.

4.35.6.3 Metadata / Georeferencing

(Starting with GDAL 1.9.0)

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.py` 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
```

4.35.6.4 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 $100 / (100 - \text{TARGET})$ 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**

4.35.7 See Also

- Implemented as `gdal/frmts/ecw/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](#)
- [GDAL ECW Build Hints](#)

4.36 EEDAI - Google Earth Engine Data API Image

Driver short name

EEDAI

New in version 2.4.

Build dependencies

libcurl

The driver supports read-only operations to access image content, using Google Earth Engine REST API.

4.36.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

4.36.2 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)

4.36.3 Open options

The following open options are available :

- **ASSET**=string: To specify the asset if not specified in the connection string.
- **BANDS**=bandname1[,bandnameX]*: 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: Size of a GDAL block, which is the minimum unit to query pixels. Default is 256.

4.36.4 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.
- 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` to YES.

4.36.5 Configuration options

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**=string: 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=string:** email to be specified together with EEDA_PRIVATE_KEY/EEDA_PRIVATE_KEY_FILE to use OAuth2 Service Account authentication.

4.36.6 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.

4.36.7 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.

4.36.8 Metadata

The driver will expose metadata reported in “properties” as dataset-level or band-level metadata.

4.36.9 Pixel encoding

By default (PIXEL_ENCODING=AUTO), the driver will request pixels in a format compatible of the number and data types of the bands. The PNG, JPEG and AUTO_JPEG_PNG can only be used with bands of type Byte.

4.36.9.1 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
```

4.36.10 See Also

- *Google Earth Engine Data API driver*

4.37 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 `gdal/frmts/raw/ehdrdataset.cpp`.

4.37.1 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.)*

4.37.2 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](#)

4.38 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 `gdal/frmts/raw/eirdataset.cpp`.

4.38.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

4.39 ELAS - Earth Resources Laboratory Applications Software

Driver short name

ELAS

Driver built-in by default

This driver is built-in by default

ELAS is an old remote sensing system still used for a variety of research projects within NASA. The ELAS format support can be found in `gdal/frmts/elas`.

4.39.1 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.*)

4.39.2 See Also

- [Short announcement of ELAS driver in GDAL](#)
- [NASA Software Used In Imaging Service](#) includes ELAS overview

4.40 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.
- **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 `gdal/frmts/raw/envidataset.cpp`.

4.40.1 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.)*

4.41 Epsilon - Wavelet compressed images

Driver short name

EPSILON

Build dependencies

epsilon 0.9.1

GDAL can read and write wavelet-compressed images through the Epsilon library. epsilon 0.9.1 is required.

The driver rely on the Open Source EPSILON library (dual LGPL/GPL licence v3). In its current state, the driver will only be able to read images with regular internal tiling.

The EPSILON driver only supports 1 band (grayscale) and 3 bands (RGB) images

This is mainly intended to be used by the *Rasterlite - Rasters in SQLite DB* driver.

4.41.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

4.41.2 Creation options

- **TARGET** Target size reduction as a percentage of the original (0-100). Defaults to 96
- **FILTER**. See EPSILON documentation or ‘gdalinfo –format EPSILON’ for full list of filter IDs. Defaults to ‘daub97lift’
- **BLOCKXSIZE**=n: Sets tile width, defaults to 256. Power of 2 between 32 and 1024
- **BLOCKYSIZE**=n: Sets tile height, defaults to 256. Power of 2 between 32 and 1024
- **MODE**=[NORMAL/OTLPF] : OTLPF is some kind of hack to reduce boundary artifacts when image is broken into several tiles. Due to mathematical constrains this method can be applied to biorthogonal filters only. Defaults to OTLPF
- **RGB_RESAMPLE**=[YES/NO] : Whether RGB buffer must be resampled to 4:2:0. Defaults to YES

4.41.3 See Also

- [EPSILON home page](#)

4.42 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 `gdal/frmts/envisat/envisatdataset.cpp`.

See Also: [Envisat Data Products](#) at ESA.

4.42.1 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.43 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.

4.43.1 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.)*

4.43.2 Creation Issues

Creation Options:

- **PIXELTYPE=value:** By setting this to SIGNEDBYTE, a new Byte file can be forced to be written as signed byte
- **PROJ=name:** 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=name:** 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=name:** 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().

4.43.3 See Also

- Implemented as `gdal/frmts/ers/ersdataset.cpp`.

4.44 EXR – Extended Dynamic Range Image File Format

New 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.

4.44.1 Creation Options

- **COMPRESS=[NONE/RLE/ZIPS/ZIP/PIZ/PXR24/B44/B44A/DWAA/DWAB]**: Compression method. Defaults to ZIP. 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**: By default tiled files will be created, unless this option is set to NO. In Create() mode, setting TILED=NO is not possible.
- **BLOCKXSIZE=n**: Sets tile width, defaults to 256.
- **BLOCKYSIZE=n**: Sets tile height, defaults to 256.

- **OVERVIEWS=YES/NO**: Whether to create overviews. Default to NO. Only compatible of CreateCopy() mode.
- **OVERVIEW_RESAMPLING=NEAR/AVERAGE/CUBIC/...**: Resampling method to use for overview creation. Defaults to CUBIC.
- **PREVIEW=YES/NO**: Whether to create a preview. Default to NO. Only compatible of 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=n**: DWA compression level. The higher, the more compressed the image will be (and the more artifacts). Defaults to 45 for OpenEXR 2.4

4.44.2 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.*)

4.45 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.

4.45.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (vsimem/, etc.)*

4.45.2 Data

4.45.2.1 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_HP.N.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: HP.N = 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_HP.N.FST, L7fppprrr_rrrYYYYMMDD_HRF.FST or L7fppprrr_rrrYYYYMMDD_HTM.FST files.

4.45.2.2 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.

4.45.3 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.

4.45.4 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**.

4.45.5 See Also

Implemented as `gdal/frmts/raw/fastdataset.cpp`.

Landsat FAST L7A format description available from http://ltpwww.gsfc.nasa.gov/IAS/htmls/l7_review.html (see ES-DIS 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

4.46 FIT – FIT

Driver short name

FIT

Driver built-in by default

This driver is built-in by default

NOTE: Implemented as `gdal/frmts/fit/fitdataset.cpp`.

4.46.1 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.)*

4.47 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 way 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.

NOTE: Implemented as `gdal/frmts/fits/fitsdataset.cpp`.

4.47.1 Notes on CFITSIO linking in GDAL

4.47.1.1 Linux

From source

Install CFITSIO headers from your distro (eg, `cfitsio-devel` on Fedora; `libcfitsio-dev` on Debian-Ubuntu), then compile GDAL as usual. CFITSIO will be automatically detected and linked.

From distros

On Fedora/CentOS install CFITSIO then GDAL with `dnf (yum)`: `cfitsio` is automatically linked.

Starting from Debian 10, Ubuntu 18.04 GDAL is packaged disabling CFITSIO link (see <https://bugs.debian.org/422537>): having GDAL linked against CFITSIO asks for recompile from source.

4.47.1.2 MacOSX

The last versions of the MacOSX packages are not linked against CFITSIO. Install CFITSIO as described in the [official documentation](#).

4.47.2 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.)*

4.48 FujiBAS – Fuji BAS Scanner Image

Driver short name

FujiBAS

Driver built-in by default

This driver is built-in by default

NOTE: Implemented as `gdal/frmts/raw/fujibasdataset.cpp`.

4.48.1 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations (vsimem/, etc.)*

4.49 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 `gdal/frmts/raw/genbindataset.cpp`.

4.49.1 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

4.50 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's name login
pwd = user password
db = Oracle server identification (database name)
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 multiples 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"
"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: like in the last two examples, the database name field could be left empty ("") and the TNSNAME will be used.

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)

4.50.1 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.*)

4.50.2 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

4.50.3 Creation Options

- **BLOCKXSIZE**: The number of pixel columns on raster block.
- **BLOCKYSIZE**: The number of pixel rows on raster block.
- **BLOCKBSIZE**: The number of bands on raster block.
- **BLOCKING**: Decline the use of blocking (NO) or request an automatic blocking size (OPTIMUM).
- **SRID**: Assign a specific EPSG projection/reference system identification to the GeoRaster.
- **INTERLEAVE**: Band interleaving mode, BAND, LINE, PIXEL (or BSQ, BIL, BIP) for band sequential, Line or Pixel interleaving.

- **DESCRIPTION:** 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:** 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())"
```

- **COMPRESS:** Compression options, JPEG-F, JP2-F, DEFLATE or NONE. The JPEG-F options is lossy, meaning that the original pixel values are changed. The JP2-F compression is lossless if JP2_QUALITY=100.
- **GENPYRAMID:** 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, BIQUADRATIC, 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:** Define the number of pyramid levels to be generated. If GENPYRAMID is not informed the resample method NN (nearest neighbor) will apply.
- **QUALITY:** Quality compression option for JPEG ranging from 0 to 100. The default is 75.
- **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=int_value** 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=int_value** Only if COMPRESS=JP2-f : Tile width. Defaults to 1024.
- **JP2_BLOCKYSIZE=int_value** Only if COMPRESS=JP2-f : Tile height. Defaults to 1024.
- **JP2_PROGRESSION=LRCP/RLCP/RPCL/PCRL/CPRL** Only if COMPRESS=JP2-f : Progression order. Defaults to LRCP.
- **NBITS:** Sub byte data type, options: 1, 2 or 4.
- **SPATIALEXTENT:** Generate Spatial Extents. The default for that options is TRUE, that means that this option only need to be informed to force the Spatial Extent to remain as NULL. If EXTENTSRID is not informed the Spatial Extent geometry will be generated with the same SRID as the GeoGeoraster object.
- **EXTENTSRID:** SRID code to be used on the Spatial Extent geometry. If the table/column has already a spatial extent, the value informed should be the same as the SRID on the Spatial Extent of the other existing GeoRaster.
- **OBJECTTABLE:** To create RDT as SDO_RASTER object inform TRUE otherwise, the default is FALSE and the RDT will be created as regular relational tables. That does not apply for Oracle version older than 11.

4.50.4 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"
```

4.50.5 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
```

4.50.6 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 schame name as long as the users is granted full access to both tables.
```

Loading:

```
% gdal_translate -of georaster input.tifgeor: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'))"
```

4.50.7 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
```

4.51 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 `gdal/frmts/gff/gff_dataset.cpp`.

4.51.1 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.52 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.

4.52.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.52.2 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.

WORLDFILE=ON: Force the generation of an associated ESRI world file (.wld).

Interlaced (progressive) GIF files can be generated by supplying the **INTERLACING=ON** option on creation.

Starting with GDAL 1.7.0, 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](#).

4.52.3 See Also

- [giflib Home Page](#)

4.53 GMT – GMT Compatible netCDF

Driver short name

GMT

Build dependencies

libnetcdf

GDAL has limited support for reading and writing netCDF *grid* files. NetCDF files that are not recognised as grids (they lack variables called dimension, and z) will be silently ignored by this driver. This driver is primarily intended to provide a mechanism for grid interchange with the [GMT](#) package. The netCDF driver should be used for more general netCDF datasets.

The units information in the file will be ignored, but x_range, and y_range information will be read to get georeferenced extents of the raster. All netCDF data types should be supported for reading.

Newly created files (with a type of GMT) will always have units of “meters” for x, y and z but the x_range, y_range and z_range should be correct. Note that netCDF does not have an unsigned byte data type, so 8bit rasters will generally need to be converted to Int16 for export to GMT.

NetCDF support in GDAL is optional, and not compiled in by default.

NOTE: Implemented as `gdal/frmts/netcdf/gmtdataset.cpp`.

See Also: [Unidata NetCDF Page](#)

4.53.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Georeferencing

This driver supports georeferencing

4.54 GPKG – GeoPackage raster

Driver short name

GPKG

Build dependencies

libsqlite3 (and any or all of PNG, JPEG, WEBP drivers)

Starting with GDAL 2.0, 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.

4.54.1 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.*)

4.54.2 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.

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.

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=1/2/3/4**: 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. Defaults to 4 (which is the always safe value).
- **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**: Whether to use the extent of actual existing tiles at the zoom level of the full resolution dataset. Defaults to NO.

- **TILE_FORMAT=PNG_JPEG/PNG/PNG8/JPEG/WEBP**: Format used to store tiles. See *Tile format* section. Only used in update mode. Defaults to PNG_JPEG.
- **QUALITY=1-100**: Quality setting for JPEG and WEBP compression. Only used in update mode. Default to 75.
- **ZLEVEL=1-9**: DEFLATE compression level for PNG tiles. Only used in update mode. Default to 6.
- **DITHER=YES/NO**: Whether to use Floyd-Steinberg dithering (for TILE_FORMAT=PNG8). Only used in update mode. Defaults to NO.

Note: open options are typically specified with “-oo name=value” syntax in most GDAL utilities, or with the GDALOpenEx() API call.

4.54.3 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

4.54.3.1 Tile formats

Tiled rasters

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.

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 ($\geq 0.1.4$) 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

Since GDAL 2.3, [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.

Warning: The [tiled gridded extension](#) initially implemented in GDAL 2.2 was not officially adopted and had been later reworked by OGC. The adopted [tiled gridded coverage data](#) has a few differences that will make GDAL 2.2 datasets not be compliant with the final extension. GDAL 2.3 can open those GDAL 2.2-generated files.

4.54.3.2 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.

4.54.3.3 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 behaviour 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 behaviour 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

4.54.3.4 Creation options

The following creation options are available:

- **RASTER_TABLE**=string. 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: If set to YES, an existing GeoPackage will not be priorly destroyed, such as to be able to add new content to it. Defaults to NO.
- **RASTER_IDENTIFIER**=string. Human-readable identifier (e.g. short name), put in the *identifier* column of the *gpkg_contents* table.
- **RASTER_DESCRIPTION**=string. Human-readable description, put in the *description* column of the *gpkg_contents* table.
- **BLOCKSIZE**=integer. Block size in width and height in pixels. Defaults to 256. Maximum supported is 4096. Should not be set when using a non-custom TILING_SCHEME.
- **BLOCKXSIZE**=integer. Block width in pixels. Defaults to 256. Maximum supported is 4096.
- **BLOCKYSIZE**=integer. Block height in pixels. Defaults to 256. Maximum supported is 4096.
- **TILE_FORMAT**=PNG/JPEG/PNG8/JPEG/WEBP/TIFF/AUTO: Format used to store tiles. See *Tile formats* section. Defaults to AUTO.
- **QUALITY**=1-100: Quality setting for JPEG and WEBP compression. Default to 75.
- **ZLEVEL**=1-9: DEFLATE compression level for PNG tiles. Default to 6.
- **DITHER**=YES/NO: Whether to use Floyd-Steinberg dithering (for TILE_FORMAT=PNG8). Defaults to NO.
- **TILING_SCHEME**=CUSTOM/GoogleCRS84Quad/GoogleMapsCompatible/InspireCRS84Quad/PseudoTMS_GlobalGeodetic/... See *Tiling schemes* section. Defaults to CUSTOM. 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.
- **ZOOM_LEVEL_STRATEGY**=AUTO/LOWER/UPPER. 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. Resampling algorithm. Only used by CreateCopy() for TILING_SCHEME different from CUSTOM. Defaults to BILINEAR.
- **PRECISION**=floating_point_value_in_vertical_units: Smallest significant value. Only used for tile gridded coverage datasets. Defaults to 1.
- **UOM**=string: (GDAL >= 2.3) Unit of Measurement. Only used for tiled gridded coverage datasets. Also set through SetUnitType()
- **FIELD_NAME**=string: (GDAL >= 2.3) Field name. Only used for tiled gridded coverage datasets. Defaults to Height.
- **QUANTITY_DEFINITION**=string: (GDAL >= 2.3) Description of the field. Only used for tiled gridded coverage datasets. Defaults to Height.
- **GRID_CELL_ENCODING**=grid-value-is-center/grid-value-is-area/ grid-value-is-corner: (GDAL >= 2.3) 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: (GDAL \geq 2.2) Set GeoPackage version (for application_id and user_version fields). In AUTO mode, this will be equivalent to 1.2 starting with GDAL 2.3.
- **ADD_GPKG_OGR_CONTENTS=**YES/NO: (GDAL \geq 2.2) Defines whether to add a gpkg_ogr_contents table to keep feature count for vector layers, and associated triggers. Defaults to YES.

4.54.4 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)

4.54.5 Metadata

GDAL uses the standardized `gpkg_metadata`gpkg_metadata` <http://www.geopackage.org/spec/#_metadata_table>`__` and gpkg_metadata_reference`gpkg_metadata_reference` <http://www.geopackage.org/spec/#_metadata_reference_table>`__` 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.

4.54.6 Level of support of GeoPackage Extensions

(Restricted to those have a raster scope)

Table 2: Extensions

Extension name	OGC adopted extension ?	Supported by GDAL?
Zoom Other intervals	Yes	Yes, since GDAL 2.0
Tiles Encoding WebP	Yes	Yes, since GDAL 2.0
Metadata	Yes	Yes, since GDAL 1.11
WKT for Coordinate Reference Systems (WKT v2)	Yes	Partially, since GDAL 2.2. GDAL can read databases using this extension. GDAL 3.0 brings support for the WKT v2 entry.
Tiled Gridded Coverage Data	Yes	Yes, since GDAL 2.3 (GDAL 2.2 supported a preliminary version of this extension)

4.54.7 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
```

4.54.8 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

4.54.9 Other notes

Development of raster support in the GeoPackage driver was financially supported by [Safe Software](#).

4.55 GRASS Raster Format

Driver short name

GRASS

Build dependencies

libgrass

GDAL optionally supports reading of existing GRASS raster maps or imagery groups, but not writing or export. The support for GRASS raster format is determined when the library is configured, and requires libgrass to be pre-installed (see Notes below).

GRASS raster maps/imagery groups can be selected in several ways.

1. The full path to the `cellhd` file can be specified. This is not a relative path, or at least it must contain all the path components within the GRASS database including the database root itself. The following example opens the raster map “proj_tm” within the GRASS mapset “PERMANENT” of the GRASS location “proj_tm” in the GRASS database located at `/u/data/grassdb`.

For example:

```
gdalinfo /u/data/grassdb/proj_tm/PERMANENT/cellhd/proj_tm
```

2. The full path to the directory containing information about an imagery group (or the REF file within it) can be specified to refer to the whole group as a single dataset. The following examples do the same thing.

For example:

```
gdalinfo /usr2/data/grassdb/imagery/raw/group/testmff/REF
gdalinfo /usr2/data/grassdb/imagery/raw/group/testmff
```

3. If there is a correct `.grassrc5` (GRASS 5), `.grassrc6` (GRASS 6) `.grass7/rc` (GRASS 7) setup file in the users home directory then raster maps or imagery groups may be opened just by the cell name. This only works for raster maps or imagery groups in the current GRASS location and mapset as defined in the GRASS setup file.

The following features are supported by the GDAL/GRASS link.

- Up to 256 entries from raster colormaps are read (0-255).
- Compressed and uncompressed integer (CELL), floating point (FCELL) and double precision (DCELL) raster maps are all supported. Integer raster maps are classified with a band type of “Byte” if the 1-byte per pixel format is used, or “UInt16” if the two byte per pixel format is used. Otherwise integer raster maps are treated as “UInt32”.
- Georeferencing information is properly read from GRASS format.
- An attempt is made to translate coordinate systems, but some conversions may be flawed, in particular in handling of datums and units.

4.55.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

4.55.2 Notes on driver variations

For GRASS 5.7 Radim Blazek has moved the driver to using the GRASS shared libraries directly instead of using libgrass. Currently (GDAL 1.2.2 and later) both version of the driver are available and can be configured using “-with-libgrass” for the libgrass variant or “-with-grass=<dir>” for the new GRASS 5.7+ library based version. The GRASS 5.7+ driver version is currently not supporting coordinate system access, though it is hoped that will be corrected at some point.

4.55.3 See Also

- [GRASS GIS home page](#)
- [libgrass page](#)

4.56 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 `gdal/frmts/aaigrid/aaigriddataset.cpp`.

4.56.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.57 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.

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 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 was added in GDAL 2.3.0 and is the [Discipline](#) field of the Section 0 of the message.

GRIB_IDS was added in GDAL 2.3.0 and is the [identification section](#) / Section 1 of the message.

GRIB_PDS_TEMPLATE_ASSEMBLED_VALUES was added in GDAL 2.3.0, and 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

4.57.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

4.57.2 Configuration options

This paragraph lists the configuration options that can be set to alter the default behaviour of the GRIB driver.

- **GRIB_NORMALIZE_UNITS=YES/NO** : (GDAL >= 1.9.0) Default 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 degree Kelvin (°K).

4.57.3 GRIB2 write support

GRIB2 write support is available since GDAL 2.3.0, 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`

4.57.3.1 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.

- **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=string**. 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` meta-data 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 will be 8.
- **IDS_CENTER=integer**. Identification of originating/generating center, according to [Table 0](#). Defaults to 255/Missing

- **IDS_SUBCENTER**=integer. Identification of originating/generating center, according to [Table C](#). Defaults to 65535/Missing
- **IDS_MASTER_TABLE**=integer. GRIB master tables version number, according to [Table 1.0](#). Defaults to 2
- **IDS_SIGNF_REF_TIME**=integer. Significance of reference time, according to [Table 1.2](#). Defaults to 0/Analysis
- **IDS_REF_TIME**=datetime as YYYY-MM-DD[THH:MM:SSZ]. Reference time. Defaults to 1970-01-01T00:00:00Z
- **IDS_PROD_STATUS**=integer. Production status of processed data, according to [Table 1.3](#). Defaults to 255/Missing
- **IDS_TYPE**=integer. Type of processed data, according to [Table 1.4](#). Defaults to 255/Missing
- **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**=string. 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**=string. 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.

4.57.3.2 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**, **JPEG200_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: Choice of the [Data representation template number](#). Defaults to AUTO.
 - 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**=integer between 1 to 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_value. Input values are multiplied by $10^{\text{DECIMAL_SCALE_FACTOR}}$ before integer encoding (and automatically divided 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. Only used for **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. Defaults to 0
- **COMPRESSION_RATIO**=integer_value between 1 and 100. 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 respect 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.

4.57.3.3 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 behaviour by setting the **INPUT_UNIT** creation option to C (for Celsius) or K (for Kelvin). The default is C.

4.57.3.4 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.

4.57.3.5 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"
```

4.57.4 See Also:

- NOAA NWS NDFD “degrib” GRIB2 Decoder
- NOAA NWS NCEP g2clib grib decoding library
- WMO GRIB1 Format Documents
- NCEP WMO GRIB2 Documentation

4.57.5 Credits

Support for GRIB2 write capabilities has been funded by Meteorological Service of Canada.

4.58 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 `gdal/frmts/gsg/gs7bgdataset.cpp`.

4.58.1 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.*)

4.59 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 (including create, delete, and copy). Currently the associated formats for color, metadata, and shapes are not supported.

NOTE: Implemented as `gdal/frmts/gsg/gsagdataset.cpp`.

4.59.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.60 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.

NOTE: Implemented as `gdal/frmts/gsg/gsbgdataset.cpp`.

4.60.1 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

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.61 GSC – GSC Geogrid

Driver short name

GSC

Driver built-in by default

This driver is built-in by default

NOTE: Implemented as `gdal/frmts/raw/gscdataset.cpp`.

4.61.1 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations* (*vsimem*/, etc.)

4.62 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.

4.62.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Georeferencing

This driver supports georeferencing

4.62.2 Creation options

- **COMPRESS=method** Set the GTA compression method: NONE (default) or one of BZIP2, XZ, ZLIB, ZLIB1, ZLIB2, ZLIB3, ZLIB4, ZLIB5, ZLIB6, ZLIB7, ZLIB8, ZLIB9.

4.62.3 See Also

- [GTA home page](#)

4.63 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.

4.63.1 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.*)

4.63.2 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.

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 most priority): PAM (Persistent Auxiliary metadata) .aux.xml sidecar file, INTERNAL (GeoTIFF keys and tags), TABFILE (.tab), WORLDFILE (.tfw, .tifw/.tiffw or .wld).

Starting with GDAL 2.2, 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, NONE. First mentioned sources are the most priority over the next 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.

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.

4.63.3 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 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 behaviour 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).

4.63.4 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.

The block size (tile width and height) used for overviews (internal or external) can be specified by setting the `GDAL_TIFF_OVR_BLOCKSIZE` environment variable to a power-of-two value between 64 and 4096. The default is 128, or starting with GDAL 3.1 to use the same block size as the full-resolution dataset if possible (i.e. block height and width are equal, a power-of-two, and between 64 and 4096).

4.63.5 Overviews and nodata masks

The following configurations can be encountered depending if overviews and nodata masks are internal or not.

There are two well supported configurations:

- Internal overviews, internal nodata mask: If a GeoTIFF file has a internal transparency mask (and the `GDAL_TIFF_INTERNAL_MASK` environment variable is not set to NO) 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)

For the two other mixed configurations, behaviour is less obvious:

- Internal overviews, external nodata mask: when running `BuildOverviews()` in update mode on the `.tif` file, only the overviews of the main bands will be generated. Overviews of the external `.msk` file must be explicitly generated by running `BuildOverviews()` on the `.msk`.
- 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, `BuildOverview()` in update mode means “`gdaladdo the.tif`” (without `-ro`). Whereas `BuildOverviews()` in read-only mode means “`gdaladdo -ro the.tif`”.

4.63.6 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) (GDAL >= 2.3)
- **TIFF_RSID**: This tag specifies a File Universal Unique Identifier, or RSID, according to DMF definition (GeoTIFF DGIWG) (GDAL >= 2.3)

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.

Starting with GDAL 1.9.0, XMP metadata can be extracted from the file, and will be reported as XML raw content in the xml:XMP metadata domain.

Starting with GDAL 1.10, EXIF metadata can be extracted from the file, and will be reported in the EXIF metadata domain.

4.63.7 Color Profile Metadata

Starting with GDAL 1.11, 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,l” format for red primary.)
- SOURCE_PRIMARIES_GREEN (xyY in “x,y,l” format for green primary)
- SOURCE_PRIMARIES_BLUE (xyY in “x,y,l” format for blue primary)
- SOURCE_WHITEPOINT (xyY in “x,y,l” 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.

4.63.8 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.

4.63.9 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.

Starting with GDAL 2.2, 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).

Starting with GDAL 2.2, 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 of 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.

4.63.10 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 behaviour 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
```

4.63.11 Open options

- **NUM_THREADS=number_of_threads/ALL_CPUS:** (From GDAL 2.1) Enable 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.
- **GEOREF_SOURCES=string:** (GDAL > 2.2) Define which georeferencing sources are allowed and their priority order. See *Georeferencing* paragraph.
- **SPARSE_OK=TRUE/FALSE** ((GDAL > 2.2): 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. The default is FALSE.

4.63.12 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. Starting with GDAL 2.2, non-standard (regarding to the intrinsic 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.

You may want to read hints to [generate and read cloud optimized GeoTIFF files](#)

4.63.12.1 Creation Options

- **TFW=YES:** Force the generation of an associated ESRI world file (.tfw). See the [World Files](#) page for details.
- **RPB=YES:** 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:** (GDAL >=2.0) Force the generation of an associated _RPC.TXT file to describe RPC (Rational Polynomial Coefficients), if RPC information is available.
- **INTERLEAVE=[BAND,PIXEL]:** By default TIFF files with pixel interleaving (PLANARCONFIG_CONTIG in TIFF terminology) are created. These are slightly less efficient than BAND interleaving for some purposes, but some applications only support pixel interleaved TIFF files.
- **TILED=YES:** By default striped TIFF files are created. This option can be used to force creation of tiled TIFF files.
- **BLOCKXSIZE=n:** Sets tile width, defaults to 256.
- **BLOCKYSIZE=n:** Set tile or strip height. Tile height defaults to 256, strip height defaults to a value such that one strip is 8K or less.
- **NBITS=n:** 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. From GDAL 1.6.0, values of n=9...15 (UInt16 type) and n=17...31 (UInt32 type) are also accepted. From GDAL 2.2, 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_...]** Set the compression to use. JPEG should generally only be used with Byte data (8 bit per channel). But starting with GDAL 1.7.0 and provided that 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). See the ["8 and 12 bit JPEG in TIFF"](#) wiki page for more details. The CCITT 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 since GDAL 2.3 when using internal libtiff and if GDAL built against libzstd >=1.0, or if built against external libtiff with zstd support. LERC/LERC_DEFLATE/LERC_ZSTD are available since GDAL 2.4 when using internal libtiff (and for LERC_ZSTD, see above mentioned conditions). None is the default.
- **NUM_THREADS=number_of_threads/ALL_CPUS:** (From GDAL 2.1) Enable multi-threaded compression by specifying the number of worker threads. Worth for slow compressions such as DEFLATE or LZMA. Will be ignored for JPEG. Default is compression in the main thread.

- **PREDICTOR=[1/2/3]**: 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.
- **DISCARD_LSB=nbits or nbits_band1,nbits_band2,...nbits_bandN**: (GDAL >= 2.0) 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** (From GDAL 1.6.0): 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. Starting with GDAL 2.2, SPARSE_OK=TRUE is also supported through the CreateCopy() interface. Starting with GDAL 2.2, 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]**: Set the JPEG quality when using JPEG compression. A value of 100 is best quality (least compression), and 1 is worst quality (best compression). The default is 75.
- **JPEGTABLESMODE=0/1/2/3**: (From GDAL 2.0) Configure how and where JPEG quantization and Huffman tables are written in the TIFF JpegTables tag and strip/tile. Default to 1.
 - 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]**: Set the level of compression when using DEFLATE compression (or LERC_DEFLATE). A value of 9 is best, and 1 is least compression. The default is 6.
- **ZSTD_LEVEL=[1-22]**: 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. The default is 9.
- **MAX_Z_ERROR=threshold**: Set the maximum error threshold on values for LERC/LERC_DEFLATE/LERC_ZSTD compression. The default is 0 (lossless).
- **WEBP_LEVEL=[1-100]**: 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). The default is 75.
- **WEBP_LOSSLESS=True/False**: (GDAL >= 2.4.0 and 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.
- **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). For GDAL < 1.10, only the YES value is supported, and it is then assumed as being PREMULTIPLIED alpha (ASSOCALPHA in TIFF). Starting with GDAL 1.10, YES is an alias for NON-PREMULTIPLIED alpha, and the other values can be used.
- **PROFILE=[GDALGeoTIFF/GeoTIFF/BASELINE]**: 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:** 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 know 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.
- **COPY_SRC_OVERVIEWS=[YES/NO]:** (`CreateCopy()` only) By setting this to `YES` (default is `NO`), 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`, 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.
- **GEOTIFF_KEYS_FLAVOR=[STANDARD/ESRI_PE]:** (GDAL \geq 2.1.0) 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 of 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 \geq 3.1.0) 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.

4.63.12.2 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.

4.63.12.3 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

4.63.12.4 Streaming operations

Starting with GDAL 2.0, 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.

4.63.12.5 Configuration options

This paragraph lists the configuration options that can be set to alter the default behaviour of the GTiff driver.

- `GTIFF_IGNORE_READ_ERRORS` : Can be set to TRUE to avoid turning libtiff errors into GDAL errors. Can help reading partially corrupted TIFF files
- `ESRI_XML_PAM` : Can be set to TRUE to force metadata in the xml:ESRI domain to be written to PAM.
- `JPEG_QUALITY_OVERVIEW` : Integer between 0 and 100. Default value : 75. Quality of JPEG compressed overviews, either internal or external.
- `GDAL_TIFF_INTERNAL_MASK` : See *Internal nodata masks* section. Default value : FALSE.
- `GDAL_TIFF_INTERNAL_MASK_TO_8BIT` : See *Internal nodata masks* section. Default value : TRUE
- `USE_RRD` : Can be set to TRUE to force external overviews in the RRD format. Default value : FALSE
- `TIFF_USE_OVR` : Can be set to TRUE to force external overviews in the GeoTIFF (.ovr) format. Default value : FALSE
- `GTIFF_POINT_GEO_IGNORE` : Can be set to TRUE to revert back to the behaviour of GDAL < 1.8.0 regarding how `PixelsPoint` is interpreted w.r.t geotransform. See rfc-33 for more details. Default value : FALSE
- `GTIFF_REPORT_COMPD_CS` : 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` : Can be set to FALSE to avoid all-in-one-strip files being presented as having. Default value : TRUE
- `GDAL_TIFF_OVR_BLOCKSIZE` : See *Overviews* section.
- `GTIFF_LINEAR_UNITS` : 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:` (GDAL >= 2.0) To decide if an approximate geotransform is acceptable when reading a .tab file. Default value: NO
- `GTIFF_DIRECT_IO =YES/NO:` (GDAL >= 2.0) Can be set to YES to use specialized `RasterIO()` implementations when reading un-compressed TIFF files (un-tiled only in GDAL 2.0, both un-tiled and tiled in GDAL 2.1) 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). Default value:NO
- `GTIFF_VIRTUAL_MEM_IO =YES/NO/IF_ENOUGH_RAM:` (GDAL >= 2.0) 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, starting with GDAL 2.1, 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. Default value:NO. 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_GEOREF_SOURCES =comma-separated list with one or several of PAM, INTERNAL, TABFILE or WORLDFILE. (GDAL >= 2.2). See [Georeferencing](#) paragraph.
- GDAL_NUM_THREADS =number_of_threads/ALL_CPUS: (GDAL >= 2.1) Enable 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, ...).
- GTIFF_WRITE_TOWGS84 =AUTO/YES/NO: (GDAL >= 3.0.3). When set to AUTO, a Geog-TOWGS84GeoKey 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.

4.63.13 See Also

- [GeoTIFF Information Page](#)
- [libtiff Page](#)
- [Details on BigTIFF file format](#)
- [COG – Cloud Optimized GeoTIFF generator](#) driver

4.63.13.1 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 `.tifw`, `.tifw`, `.jgw` or other suffixes depending on the image file it comes with.

4.64 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. From GDAL 1.8.0, 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 `gdal/frmts/gxf/gxfdataset.cpp`.

4.64.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.65 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.

4.65.1 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

4.65.2 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
```

(continues on next page)

(continued from previous page)

```
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)
  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
```

(continues on next page)

(continued from previous page)

```

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).

4.65.3 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 behaviour by setting the **GEOL_AS_GCPS** environment variable to **PARTIAL** (default), **NONE**, or **FULL**.

4.65.4 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:

- **RANK=n**: Create **n**-dimensional SDS. Currently only 2D and 3D datasets supported. By default a 3-dimensional dataset will be created.

4.65.5 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.

4.65.6 Multidimensional API support

New in version 3.1.

The HDF4 driver supports the *Multidimensional Raster Data Model* for reading operations.

4.65.7 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).

4.65.8 See Also

- Implemented as `gdal/frmts/hdf4/hdf4dataset.cpp` and `gdal/frmts/hdf4/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](#)

4.66 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.

4.66.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

New in version 2.4:

Supports VirtualIO

This driver supports *virtual I/O operations (vsimem/, etc.)*

4.66.2 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).

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
```

(continues on next page)

(continued from previous page)

```

SUBDATASET_10_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_O3/Data_Fields/
→ 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/GroundPixelQualityFlags (16-bit unsigned integer)

```

(continues on next page)

(continued from previous page)

```

SUBDATASET_24_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-
↳2005m0625t035355.he5"://HDFEOS/SWATHS/OMI_Column_Amount_03/Geolocation_Fields/
↳Latitude
SUBDATASET_24_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_03/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_03/Geolocation_Fields/
↳Longitude
SUBDATASET_25_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_03/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_03/Geolocation_Fields/
↳RelativeAzimuthAngle
SUBDATASET_26_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_03/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_03/Geolocation_Fields/
↳SolarAzimuthAngle
SUBDATASET_27_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_03/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_03/Geolocation_Fields/
↳SolarZenithAngle
SUBDATASET_28_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_03/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_03/Geolocation_Fields/
↳TerrainHeight
SUBDATASET_29_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_03/Geolocation_
↳Fields/TerrainHeight (16-bit integer)
SUBDATASET_30_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-
↳2005m0625t035355.he5"://HDFEOS/SWATHS/OMI_Column_Amount_03/Geolocation_Fields/
↳ViewingAzimuthAngle
SUBDATASET_30_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_03/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_03/Geolocation_Fields/
↳ViewingZenithAngle
SUBDATASET_31_DESC=[1496x60] //HDFEOS/SWATHS/OMI_Column_Amount_03/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",

```

(continues on next page)

(continued from previous page)

```

    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 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).

4.66.3 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 OMT03**)

4.66.4 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

4.66.5 Multidimensional API support

New in version 3.1.

The HDF5 driver supports the *Multidimensional Raster Data Model* for reading operations.

4.66.6 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).

4.66.7 See Also

Implemented as `gdal/frmts/hdf5/hdf5dataset.cpp` and `gdal/frmts/hdf5/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:

- OMT03: OMI/Aura Ozone (O3) Total Column 1-Orbit L2 Swath 13x24km V003

4.67 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.

4.67.1 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.*)

4.67.2 Creation options

- **COMPRESS=YES/NO** : whether the file must be compressed with GZip or no. Defaults to NO
- **BLOCKSIZE=block_size** : internal tile size. Must be ≥ 8 . Defaults to 256
- **VERTICAL_PRECISION=vertical_precision** : Must be > 0 . Defaults to 0.01

Increasing the vertical precision decreases the file size, especially with COMPRESS=YES, but at the loss of accuracy.

4.67.3 See also

- [Specification of HF2/HFZ format](#)
- [Specification of HF2 extended header blocks](#)

4.68 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 support for reading and writing.

Metadata reading and writing is supported at the dataset level, and for bands, but this is GDAL specific metadata - not metadata in an Imagine recognized form. The metadata is stored in a table called GDAL_MetaData with each column being a metadata item. The title is the key and the row 1 value is the value.

4.68.1 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.)*

4.68.2 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:

- **BLOCKSIZE=blocksize**: Tile width/height (32-2048). Default=64
- **USE_SPILL=YES**: Force the generation of a spill file (by default spill file created for images larger 2GiB only). Default=NO
- **COMPRESSED=YES**: Create file as compressed. Use of spill file disables compression. Default=NO
- **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.
- **AUX=YES**: To create a .aux file. Default=NO
- **IGNOREUTM=YES** : Ignore UTM when selecting coordinate system - will use Transverse Mercator. Only used for Create() method. Default=NO
- **STATISTICS=YES** : To generate statistics and a histogram. Default=NO
- **DEPENDENT_FILE=filename** : Name of dependent file (must not have absolute path). Optional

- **FORCETOPESTRING=YES**: Force use of ArcGIS PE String in file instead of Imagine coordinate system format. In some cases this improves ArcGIS coordinate system compatibility.

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)

4.68.3 Configuration Options

Currently three [runtime 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** : (GDAL >= 1.11) 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 >= 2.3) The block size (tile width/height) used for overviews can be specified by setting the **GDAL_HFA_OVR_BLOCKSIZE** configuration option to a power- of-two value between 32 and 2048. The default value is 64.

4.68.4 See Also

- Implemented as `gdal/frmts/hfa/hfadataset.cpp`.
- More information, and other tools are available on the [Imagine \(.img\) Reader](#) page as saved by [archive.org](#).
- [Erdas.com](#)

4.69 IDA – Image Display and Analysis

Driver short name

IDA

Driver built-in by default

This driver is built-in by default

GDAL supports reading and writing IDA images with some limitations. IDA images are the image format of WinDisp 4. The files are always one band only of 8bit data. IDA files often have the extension .img though that is not required.

Projection and georeferencing information is read though some projections (i.e. Meteosat, and Hammer-Aitoff) are not supported. When writing IDA files the projection must have a false easting and false northing of zero. The support coordinate systems in IDA are Geographic, Lambert Conformal Conic, Lambert Azimuth Equal Area, Albers Equal-Area Conic and Goodes Homolosine.

IDA files typically contain values scaled to 8bit via a slope and offset. These are returned as the slope and offset values of the bands and they must be used if the data is to be rescaled to original raw values for analysis.

NOTE: Implemented as `gdal/frmts/raw/idadataset.cpp`.

See Also: [WinDisp](#)

4.69.1 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.*)

4.70 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

4.70.1 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.)*

4.70.2 See Also

- Implemented as `gdal/frmts/idrisi/idrisiraster.cpp`.
- www.idrisi.com

4.71 IGNFHeightASCIIGrid – IGN-France height correction ASCII grids

Driver short name

IGNFHeightASCIIGrid

New in version 2.4.

Driver built-in by default

This driver is built-in by default

Supports reading IGN-France height correction ASCII grids (.txt, .mnt, .gra).

See also:

- [Format description \(in French\)](#)
- [Height correction grids](#)

4.71.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

4.72 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 GeoReferenceCorner 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 `gdal/frmts/ilwis`.

See Also: <http://www.itc.nl/ilwis/default.asp> .

4.72.1 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.)*

4.73 INGR – Intergraph Raster Format

Driver short name

INGR

Driver built-in by default

This driver is built-in by default

This format is supported for read and writes access.

The Intergraph Raster File Format was the native file format used by Intergraph software applications to store raster data. It is manifested in several internal data formats.

4.73.1 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.)*

4.73.2 Reading INGR Files

Those are the data formats that the INGR driver supports for reading:

- 2 - Byte Integer
- 3 - Word Integer
- 4 - Integers 32 bit
- 5 - Floating Point 32 bit
- 6 - Floating Point 64 bit
- 9 - Run Length Encoded
- 10 - Run Length Encoded Color
- 24 - CCITT Group 4
- 27 - Adaptive RGB
- 28 - Uncompressed 24 bit
- 29 - Adaptive Gray Scale
- 30 - JPEG GRAY
- 31 - JPEG RGB
- 32 - JPEG CMYK
- 65 - Tiled
- 67 - Continuous Tone

The format “65 - Tiled” is not a format; it is just an indication that the file is tiled. In this case the tile header contains the real data format code that could be any of the above formats. The INGR driver can read tiled and untiled instance of any of the supported data formats.

4.73.3 Writing INGR Files

Those are the data formats that the INGR driver supports for writing:

- 2 - Byte Integer
- 3 - Word Integers
- 4 - Integers 32Bit
- 5 - Floating Point 32Bit
- 6 - Floating Point 64Bit

Type 9 RLE bitonal compression is used when outputting “.rle” file. Other file types are uncompressed.

Note that writing in that format is not encouraged.

4.73.4 File Extension

The following is a partial listing of INGR file extensions:

.cot	8-bit grayscale or color table data
.ctc	8-bit grayscale using PackBits-type compression (uncommon)
.rgb	24-bit color and grayscale (uncompressed and PackBits-type compression)
.ctb	8-bit color table data (uncompressed or run-length encoded)
.grd	8, 16 and 32 bit elevation data
.crl	8 or 16 bit, run-length compressed grayscale or color table data
.tpe	8 or 16 bit, run-length compressed grayscale or color table data
.lsr	8 or 16 bit, run-length compressed grayscale or color table data
.rle	1-bit run-length compressed data (16-bit runs)
.cit	CCITT G3 or G4 1-bit data
.g3	CCITT G3 1-bit data
.g4	CCITT G4 1-bit data
.tg4	CCITT G4 1-bit data (tiled)
.cmp	JPEG grayscale, RGB, or CMYK
.jpg	JPEG grayscale, RGB, or CMYK

Source: <http://www.oreilly.com/www/centers/gff/formats/ingr/index.htm>

The INGR driver does not require any especial file extension in order to identify or create an INGR file.

4.73.5 Georeference

The INGR driver does not support reading or writing georeference information. The reason for that is because there is no universal way of storing georeferencing in INGR files. It could have georeference stored in a accompanying .dgn file or in application specific data storage inside the file itself.

4.73.6 Metadata

The following creation option and bandset metadata is available.

- RESOLUTION: This is the DPI (dots per inch). Microns not supported.

4.73.7 See Also

For more information:

- Implemented as `gdal/frmts/ingr/intergraphraster.cpp`.
- www.intergraph.com
- <http://www.oreilly.com/www/centers/gff/formats/ingr/index.htm>
- File specification: <ftp://ftp.intergraph.com/pub/bbs/scan/note/rffrgps.zip/>

4.74 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 `gdal/frmts/iris/irisdataset.cpp`.

4.74.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.75 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 `gdal/frmts/raw/iscedataset.cpp`.

4.75.1 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.*)

4.76 ISG – International Service for the Geoid

New 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 http://www.isgeoid.polimi.it/Geoid/ISG_format_20160121.pdf

NOTE: Implemented as `gdal/frmts/aaigrid/aaigriddataset.cpp`.

4.76.1 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.*)

4.77 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 and write 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.

4.77.1 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.)*

4.77.2 Creation Issues

Currently the ISIS2 writer writes a very minimal header with only the image structure information. No coordinate system, georeferencing or other metadata is captured.

4.77.2.1 Creation Options

- **LABELING_METHOD=ATTACHED/DETACHED:** Determines whether the header labeling should be in the same file as the imagery (the default - ATTACHED) or in a separate file (DETACHED).
- **IMAGE_EXTENSION=extension:** Set the extension used for detached image files, defaults to “cub”. Only used if LABELING_METHOD=DETACHED.

4.77.3 See Also

- Implemented as `gdal/frmts/pds/isis2dataset.cpp`.
- *PDS – Planetary Data System v3* driver
- *ISIS3 – USGS Astrogeology ISIS Cube (Version 3)* driver

4.78 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. Since GDAL 2.2, 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.

Starting with GDAL 2.2, 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.

4.78.1 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.*)

4.78.2 Metadata

Starting with GDAL 2.2, the ISIS3 label can be retrieved as JSON-serialized content in the json:ISIS3 metadata domain.

For example:

```
$ python
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
      }
    },
    "Instrument":{
      "_type":"group",
      "TargetName":"Mars"
    },
    "BandBin":{
      "_type":"group",
      "Center":1.000000,
      "OriginalBand":1
    },
    "Mapping":{
      "_type":"group",
      "ProjectionName":"Equiarectangular",
      "CenterLongitude":184.412994,
      "TargetName":"Mars",
      "EquatorialRadius":{
        "value":3396190.000000,
        "unit":"meters"
      },
      "PolarRadius":{
```

(continues on next page)

(continued from previous page)

```

        "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,
    "Bytes":957,
    "^History":"r0200357_10m_Jul20_o_i3_detatched.History.IsisCube"
},
"OriginalLabel":{
    "_type":"object",
    "Name":"IsisCube",
    "StartByte":1,
    "Bytes":2482,
    "^OriginalLabel":"r0200357_10m_Jul20_o_i3_detatched.OriginalLabel.IsisCube"
}
}

```

or

```
$ gdalinfo -json ../autotest/gdrivers/data/isis3_detatched.lbl -mdd all
```

On creation, a source template label can be passed to the SetMetadata() interface in the “json:ISIS3” metadata domain.

4.78.3 Creation support

Starting with GDAL 2.2, 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.

The available creation options are:

- **DATA_LOCATION**=LABEL/EXTERNAL/GEOTIFF. 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. 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**=string. Comma separated list of KEY=VALUE tuples to forward to the GeoTIFF driver. e.g. `GEOTIFF_OPTIONS=COMPRESS=LZW`.
- **EXTERNAL_FILENAME**=filename. Override default external filename. Only for `DATA_LOCATION=EXTERNAL` or `GEOTIFF`.
- **TILED**=YES/NO. Whether the pixel data should be tiled. Default is NO (ie band sequential organization).
- **BLOCKXSIZE**=int_value. Tile width in pixels. Only used if `TILED=YES`. Defaults to 256.
- **BLOCKYSIZE**=int_value. Tile height in pixels. Only used if `TILED=YES`. Defaults to 256.
- **COMMENT**=string. Comment to add into the label.
- **LATITUDE_TYPE**=Planetocentric/Planetographic. Value of `Mapping.LatitudeType`. Defaults to Planetocentric. If specified, and `USE_SRC_MAPPING` is in effect, this will be taken into account to override the source `LatitudeType`.
- **LONGITUDE_DIRECTION**=PositiveEast/PositiveWest. Value of `Mapping.LongitudeDirection`. Defaults to PositiveEast. If specified, and `USE_SRC_MAPPING` is in effect, this will be taken into account to override the source `LongitudeDirection`.
- **TARGET_NAME**=string. 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. Whether to force longitudes in the [0, 360] range. Defaults to NO.
- **WRITE_BOUNDING_DEGREES**=YES/NO. Whether to write Min/MaximumLong/ Latitude values. Defaults to YES.
- **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. Whether to use source label in ISIS3 to ISIS3 conversions. Defaults to YES.
- **USE_SRC_MAPPING**=YES/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. Whether to use the content pointed by the source History object in ISIS3 to ISIS3 conversions, and write it to the new dataset. Defaults to YES. 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.** Whether to add GDAL specific history in the content pointed by the History object in ISIS3 to ISIS3 conversions. Defaults to YES. 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=string.** 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).

4.78.4 Examples

How to create a copy of a source ISIS3 dataset to another ISIS3 dataset while modifying a parameter of Isis-Cube.Mapping group, by using GDAL Python :

```
import json
from osgeo import gdal

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
```

4.78.5 See Also

- Implemented as `gdal/frmts/pds/isis3dataset.cpp`.
- *GDAL PDS Driver*
- *GDAL ISIS2 Driver*

4.79 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 `gdal/frmts/jdem/jdemdataset.cpp`.

See Also: [Geographic Survey Institute \(GSI\) Web Site](#).

4.79.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.80 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.

4.80.1 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.*)

4.80.2 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.

4.80.3 History

- v3.x - Last release, 2006
- v4.x - Last release, 2012
- v5.x - Active development, 2013 - current

4.80.4 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 most priority): PAM, GeoJP2, GMLJP2, WORLDFILE.

Starting with GDAL 2.2, 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. First mentioned sources are the most priority over the next 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.

4.80.5 Option Options

(GDAL >= 2.0) The following open option is available:

- **1BIT_ALPHA_PROMOTION=YES/NO**: Whether a 1-bit alpha channel should be promoted to 8-bit. Defaults to YES.
- **GEOREF_SOURCES=string**: (GDAL > 2.2) Define which georeferencing sources are allowed and their priority order. See [Georeferencing](#) paragraph.

4.80.6 Creation Options:

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**: 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=name**: Name of the ECW projection string to use. Common examples are NUTM11, or GEODETIC.

- **DATUM=name**: Name of the ECW datum string to use. Common examples are WGS84 or NAD83.
- **GMLJP2=YES/NO**: 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. Defaults to YES.
- **GMLJP2V2_DEF=filename**: (Starting with GDAL 2.0) 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**: Indicates whether a UUID/GeoTIFF box conforming to the GeoJP2 (GeoTIFF in JPEG2000) specification should be included in the file. Defaults to YES.
- **PROFILE=profile**: One of BASELINE_0, BASELINE_1, BASELINE_2, NPJE or EPJE. Review the ECW SDK documentation for details on profile meanings.
- **PROGRESSION=LRCP/RLCP/RPCL**: Set the progression order with which the JPEG2000 codestream is written. (Default, RPCL)
- **CODESTREAM_ONLY=YES/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. (Default, NO)
- **LEVELS=n**: Resolution levels in pyramid (by default so many that the size of the smallest thumbnail image is 64x64 pixels at maximum)
- **LAYERS=n**: Quality layers (default, 1)
- **PRECINCT_WIDTH=n**: Precinct Width (default, 64)
- **PRECINCT_HEIGHT=n**: Precinct Height (default 64)
- **TILE_WIDTH=n**: 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=n**: 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=n**: The number of quality layers to decode
- **DECOMPRESS_RECONSTRUCTION_PARAMETER=n**: IRREVERSIBLE_9x7 or REVERSIBLE_5x3
- **WRITE_METADATA=YES/NO**: (GDAL >= 2.0) 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:** (GDAL >= 2.0) (Only if WRITE_METADATA=YES) Whether only metadata from the main domain should be written. Defaults to NO.

“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.”

4.80.7 Configuration Options

The ERDAS ECW/JP2 SDK supports a variety of [runtime configuration options](#) to control various features. Most of these are exposed as GDAL configuration options. 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:** This may be set to FALSE to disable dithering when decompressing ECW files. Defaults to TRUE.
- **ECW_FORCE_FILE_REOPEN=TRUE/FALSE:** This may be set to TRUE to force open a file handle for each file for each connection made. Defaults to FALSE.
- **ECW_CACHE_MAXOPEN=number:** The maximum number of files to keep open for ECW file handle caching. Defaults to unlimited.
- **ECW_AUTOGEN_J2I=TRUE/FALSE:** Controls whether .j2i index files should be created when opening jpeg2000 files. Defaults to TRUE.
- **ECW_RESILIENT_DECODING=TRUE/FALSE:** Controls whether the reader should be forgiving of errors in a file, trying to return as much data as is available. Defaults to TRUE. If set to FALSE an invalid file will result in an error.

4.80.8 Metadata

Starting with GDAL 1.11.0, 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

4.80.9 See Also

- Implemented as `gdal/frmts/ecw/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](#)
- [GDAL ECW Build Hints](#)

4.81 JP2KAK – JPEG-2000 (based on Kakadu)

Driver short name

JP2KAK

Build dependencies

Kakadu library

Most forms of JPEG2000 JP2 and JPC compressed images (ISO/IEC 15444-1) can be read with GDAL using a driver based on the Kakadu library. As well, new images can be written. Existing images cannot be updated in place.

The JPEG2000 file format supports lossy and lossless compression of 8bit and 16bit images with 1 or more bands (components). Via the [GeoJP2 \(tm\)](#) mechanism, GeoTIFF style coordinate system and georeferencing information can be embedded within the JP2 file. JPEG2000 files use a substantially different format and compression mechanism than the traditional JPEG compression and JPEG JFIF format. They are distinct compression mechanisms produced by the same group. JPEG2000 is based on wavelet compression.

The JPEG2000 driver documented on this page (the JP2KAK driver) is implemented on top of the proprietary [Kakadu](#) library. This is a high quality and high performance JPEG2000 library in wide used in the geospatial and general imaging community. However, it is not free, and so normally builds of GDAL from source will not include support for this driver unless the builder purchases a license for the library and configures accordingly.

When reading images this driver will represent the bands as being Byte (8bit unsigned), 16 bit signed or 16 bit 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.

Starting with GDAL 1.9.0, XMP metadata can be extracted from JPEG2000 files, and will be stored as XML raw content in the xml:XMP metadata domain.

4.81.1 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.)*

4.81.2 Configuration Options

The JP2KAK driver supports the following [Config Options](#). These runtime options can be used to alter the behavior of the driver.

- **JP2KAK_THREADS=n**: By default an effort is made to take advantage of multi-threading on multi-core computers using default rules from the Kakadu library. 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**: This can be set to YES to turn on fussy reporting of problems with the JPEG2000 data stream. Defaults to NO.
- **JP2KAK_RESILIENT=YES/NO**: This can be set to YES to force Kakadu to maximize resilience with incorrectly created JPEG2000 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. Defaults to NO.

4.81.3 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 most priority): PAM, GeoJP2, GMLJP2, WORLDFILE.

Starting with GDAL 2.2, 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. First mentioned sources are the most priority over the next 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.

4.81.4 Option Options

(GDAL >= 2.0) The following open option is available:

- **1BIT_ALPHA_PROMOTION=YES/NO:** Whether a 1-bit alpha channel should be promoted to 8-bit. Defaults to YES.
- **GEOREF_SOURCES=string:** (GDAL > 2.2) Define which georeferencing sources are allowed and their priority order. See *Georeferencing* paragraph.

4.81.5 Creation Issues

JPEG2000 files can only be created using the CreateCopy mechanism to copy from an existing dataset.

JPEG2000 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:

- **CODEC=JP2/J2K** Codec to use. If not specified, guess based on file extension. If unknown, default to JP2
- **QUALITY=n:** 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=n:** Set the tile width to use. Defaults to 20000.
- **BLOCKYSIZE=n:** Set the tile height to use. Defaults to image height.
- **FLUSH=TRUE/FALSE:** 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. Defaults to TRUE.
- **GMLJP2=YES/NO:** 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. Defaults to YES.
- **GMLJP2V2_DEF=filename:** (Starting with GDAL 2.0) 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:** Indicates whether a UUID/GeoTIFF box conforming to the GeoJP2 (GeoTIFF in JPEG2000) specification should be included in the file. Defaults to YES.
- **LAYERS=n:** Control the number of layers produced. These are sort of like resolution layers, but not exactly. The default value is 12 and this 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 library, and are considered to be for advanced use only. Consult Kakadu documentation to better understand their meaning.

- **Corder:** Defaults to “PRCL”.
- **Cprecincts:** Defaults to “{512,512},{256,512},{128,512},{64,512},{32,512},{16,512},{8,512},{4,512},{2,512}”.
- **ORGgen_plt:** Defaults to “yes”.
- **Cmodes:** Kakadu library default used.
- **Clevels:** Kakadu library default used.
- **Rshift:** Kakadu library default used.
- **Rlevels:** Kakadu library default used.
- **Rweight:** Kakadu library default used.

4.81.6 Known Kakadu Issues

4.81.6.1 Alpha Channel Writing in v7.8

Kakadu 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)
```

4.81.6.2 kdu_get_num_processors always returns 0 for some platforms

On non-windows / non-mac installs (e.g. Linux), Kakadu 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.

4.81.7 See Also

- Implemented as `gdal/frmts/jp2kak/jp2kakdataset.cpp`.
- If you’re using a Kakadu release before v7.5, configure & compile GDAL with eg. `CXXFLAGS="-DKDU_MAJOR_VERSION=7 -DKDU_MINOR_VERSION=3 -DKDU_PATCH_VERSION=2"` for Kakadu version 7.3.2.
- Alternate *JP2OpenJPEG – JPEG2000 driver based on OpenJPEG library* driver.

4.82 JP2Lura – JPEG2000 driver based on Lurawave library

Driver short name

JP2LURA

New in version 2.2.

Build dependencies

Lurawave library

This driver is an implementation of a JPEG2000 reader/writer based on Lurawave library.

The driver uses the VSI Virtual File API, so it can read JPEG2000 compressed NITF files.

4.82.1 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.)*

4.82.2 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 most 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. First mentioned sources are the most priority over the next 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.

4.82.3 License number

The LURA_LICENSE_NUM_1 and LURA_LICENSE_NUM_2 configuration options / environment variables must be set with the 2 numbers that compose a license number.

4.82.4 Option Options

The following open option is available:

- **GEOREF_SOURCES=string**: Define which georeferencing sources are allowed and their priority order. See [Georeferencing](#) paragraph.

4.82.5 Creation Options

- **CODEC=JP2/Codestream** : JP2 will add JP2 boxes around the codestream data. The value is determined automatically from the file extension. If it is neither JP2 nor Codestream, JP2 codec is used.
- **GMLJP2=YES/NO**: 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. Defaults to YES.
- **GMLJP2V2_DEF=filename**: 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 [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**: Indicates whether a UUID/GeoTIFF box conforming to the GeoJP2 (GeoTIFF in JPEG2000) specification should be included in the file. Defaults to NO.
- **SPLIT_IEEE754=YES/NO**: Whether encoding of Float32 bands as 3 bands with values decomposed according to IEEE-754 structure: first band (1 bit, signed) with sign bit, second band (8 bits, unsigned) with exponent value and third band (23 bits, unsigned) with mantissa value. Default to NO. This is a non-standard extension to encode floating point values. By default, the sign bit and exponent will be encoded with the reversible wavelet (even with REVERSIBLE=NO), and the mantissa with the irreversible one. If specifying REVERSIBLE=YES, all 3 components will be encoded with the reversible wavelet.
- **NBITS=int_value** : Bits (precision) for sub-byte files (1-7), sub-uint16 (9-15), sub-uint32 (17-28).
- **QUALITY_STYLE=PSNR/XXSmall/XSmall/Small/Medium/Large/XLarge/XXLarge**: This property tag is used to set the quality mode to be used during lossy compression. For normal images and situations (1:1 pixel display, ~50 cm viewing distance) we recommend Small or PSNR. For quality measurement only PSNR should be used. Default is PSNR.
- **SPEED_MODE=Fast/Accurate**: This property tag is used to set the speed mode to be used during lossy compression. The following modes are defined. Default is Fast
- **RATE=int_value**. When specifying this value, the target compressed file size will be the uncompressed file size divided by RATE. In general the achieved rate will be exactly the requested size or a few bytes lower. Will force use of irreversible wavelet. Default value: 0 (maximum quality).
- **QUALITY=1 to 100** Compression to a particular quality is possible only when using the 9-7 filter with the standard expounded quantization and no regions of interest. A compression quality may be specified between 1 (low) and 100 (high). The size of the resulting JPEG2000 file will depend of the image content. Only used for irreversible compression. The compression quality cannot be used together the property RATE. Default value: 0 (maximum quality). When using this option together with SPLIT_IEEE754=YES, the sign bit and exponent

bands will have to be switched to irreversible encoding, which can lead to huge loss in the reconstructed floating-point value.

- **PRECISION=int_value** For improved efficiency, the library automatically, depending on the image depth, uses either 16 or 32 bit representation for wavelet coefficients. The precision property can be set to force the library to always use 32 bit representations. The use of 32 bit values may slightly improve image quality and the expense of speed and memory requirements. Default value: 0 (automatically select appropriate precision).
- **REVERSIBLE=YES/NO** : YES means use of reversible 5x3 integer-only filter, NO use of the irreversible DWT 9-7. Defaults to NO.
- **LEVELS=int_value** (0-16) : The number of wavelet transformation levels can be set using this property. Valid values are in the range 0 (no wavelet analysis) to 16 (very fine analysis). The memory requirements and compression time increases with the number of transformation levels. A reasonable number of transformation levels is in the 4-6 range. Default is 5.
- **QUANTIZATION_STYLE=DERIVED/EXPONDED** : This property may only be set when the irreversible filter (9_7) is used. The quantization steps can either be derived from a bases quantization step, DERIVED, or calculated for each image sub-band, EXPONDED. The EXPONDED style is recommended when using the irreversible filter. Default is EXPONDED.
- **TILEXSIZE=int_value** : Tile width. An image can be split into smaller tiles, with each tile independently compressed. The basic tile size and the offset to the first tile on the virtual compression reference grid can be set using these properties. The first tile must contain the first image pixel. The tiling of an image is recommended only for very large images. Default values: (0) One Tile containing the complete image. If the image dimension exceeds 15000x15000, it will be tiled with tiles of dimension 1024x1024.
- **TILEYSIZE=int_value** : Tile height. An image can be split into smaller tiles, with each tile independently compressed. The basic tile size and the offset to the first tile on the virtual compression reference grid can be set using these properties. The first tile must contain the first image pixel. The tiling of an image is recommended only for very large images. Default values: (0) One Tile containing the complete image. If the image dimension exceeds 15000x15000, it will be tiled with tiles of dimension 1024x1024.
- **TLM=YES/NO**: (TiLe Marker) The efficiency of decoding regions in a tiled image may be improved by " the usage of a tile length marker. Tile length markers contain the " position of each tile in a JPEG2000 codestream, enabling faster access " to tiled data. Default is NO.
- **PROGRESSION=LRCP/RLCP/RPCL/PCRL/CPRL** : The organization of the coded data in the file can be set by this property tag. The following progression orders are defined: LRCP = Quality progressive, LCP = Resolution then quality progressive, RPCL = Resolution then position progressive, PCRL = Position progressive, CPRL = Color/channel progressive. The setting LRCP (quality) is most useful when used with several layers. The PCRL (position) should be used with precincts. Defaults to LRCP.
- **JPX=YES/NO**: Whether to advertize JPX features, and add a Reader requirement box, when a GMLJP2 box is written (for GMLJP2 v2, the branding will also be "jpx "). Defaults to YES. This option should not be used unless compatibility problems with a reader occur.
- **CODEBLOCK_WIDTH=int_value**: Codeblock width: power of two value between 4 and 1024. Defaults to 64. Note that CODEBLOCK_WIDTH * CODEBLOCK_HEIGHT must not be greater than 4096. For PRO-FILE_1 compatibility, CODEBLOCK_WIDTH must not be greater than 64.
- **CODEBLOCK_HEIGHT=int_value**: Codeblock height: power of two value between 4 and 1024. Defaults to 64. Note that CODEBLOCK_WIDTH * CODEBLOCK_HEIGHT must not be greater than 4096. For PRO-FILE_1 compatibility, CODEBLOCK_HEIGHT must not be greater than 64.
- **ERROR_RESILIENCE=YES/NO**: This option improves error resilient in JPEG2000 streams or for special codecs (e.g. hardware coder) for a faster compression/ decompression. This option will increase the file size slightly when generating a code stream with the same image quality. Default is NO.

- **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:** (Only if WRITE_METADATA=YES) Whether only metadata from the main domain should be written. Defaults to NO.
- **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, ...

4.82.5.1 Lossless compression

Lossless compression can be achieved if REVERSIBLE=YES is used (and RATE is not specified).

4.82.6 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 `JP2Lura' 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.

4.82.7 Bugs

Proper support of JPEG-2000 images with Int32/UInt32/Float32-IEEE754-split on Linux 64 bits require a v2.1.00.17 or later SDK.

4.82.8 See Also

- [LuraTech JPEG-2000 SDK](#)

Other JPEG2000 GDAL drivers :

- *JP2OpenJPEG: based on Openjpeg library (open source)*
- *JPEG2000: based on Jasper library (open source)*
- *JP2ECW: based on Erdas ECW library (proprietary)*
- *JP2MRSID: based on LizardTech MrSID library (proprietary)*
- *JP2KAK: based on Kakadu library (proprietary)*

4.83 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.

4.83.1 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.*)

4.83.2 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 most priority): PAM, GeoJP2, GMLJP2, WORLDFILE.

Starting with GDAL 2.2, 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. First mentioned sources are the most priority over the next 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.

4.83.3 Creation Options

If you have the MrSID ESDK (5.x or newer), it can be used to write JPEG2000 files. The following creation options are supported.

- **WORLDFILE=YES:** to write an ESRI world file (with the extension .j2w).
- **COMPRESSION=n:** 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=[path to file]:** 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>
      8
    </layers>
    <levels>
      99
    </levels>
    <tileSize>
      0 0
    </tileSize>
    <progressionOrder>
      RPCL
    </progressionOrder>
    <codeblockSize>
      64 64
    </codeblockSize>
    <pltMarkers>
      true
    </pltMarkers>
    <wavelet97>
      false
  </Codestream>
</Jp2Profile>
```

(continues on next page)

(continued from previous page)

```
</wavelet97>
<precinctSize>
  256 256
</precinctSize>
</Codestream>
</Jp2Profile>
```

4.83.4 See Also

- Implemented as `gdal/frmts/mrsid/mrsiddataset.cpp`.
- [Extensis web site](#)

4.84 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.

Starting with GDAL 2.0, 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.

4.84.1 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.*)

4.84.2 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 most priority): PAM, GeoJP2, GMLJP2, WORLDFILE.

Starting with GDAL 2.2, 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. First mentioned sources are the most priority over the next 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.

4.84.3 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 behaviour 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-threading 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 NUM_CPUS. Its default value is 1. Starting with GDAL 2.3, this multi-threading at code-block level is automatically enabled by GDAL

Both multi-threading mechanism can be combined together.

4.84.4 Option Options

(GDAL >= 2.0) The following open option is available:

- **1BIT_ALPHA_PROMOTION=YES/NO:** Whether a 1-bit alpha channel should be promoted to 8-bit. Defaults to YES.
- **GEOREF_SOURCES=string:** (GDAL > 2.2) Define which georeferencing sources are allowed and their priority order. See [Georeferencing](#) paragraph.
- **USE_TILE_AS_BLOCK=YES/NO:** (GDAL > 2.2) Whether to always use the JPEG-2000 block size as the GDAL block size Defaults to NO. 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.

4.84.5 Creation Options

- **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**: (Starting with GDAL 1.10) 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. Defaults to YES.
- **GMLJP2V2_DEF=filename**: (Starting with GDAL 2.0) 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. Note: GDAL 2.0 and 2.1 use the older [OGC GML in JPEG2000, version 2.0](#) specification, that differ essentially by the content of the gml:domainSet, gml:rangeSet and gmlcov:rangeType elements of gmljp2:GMLJP2CoverageCollection.
- **GeoJP2=YES/NO**: (Starting with GDAL 1.10) Indicates whether a UUID/GeoTIFF box conforming to the GeoJP2 (GeoTIFF in JPEG2000) specification should be included in the file. Defaults to YES.
- **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). Starting with GDAL 2.0, 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=int_value** : Number of resolution levels. Default value is selected such the smallest overview of a tile is no bigger than 128x128.
- **BLOCKXSIZE=int_value** : Tile width. Defaults to 1024.
- **BLOCKYSIZE=int_value** : Tile height. Defaults to 1024.
- **PROGRESSION=LRCP/RLCP/RPCL/PCRL/CPRL** : Progession order. Defaults to LRCP.
- **SOP=YES/NO** : YES means generate SOP (Start Of Packet) marker segments. Defaults to NO.
- **EPH=YES/NO** : YES means generate EPH (End of Packet Header) marker segments. Defaults to NO.
- **YCBCR420=YES/NO** : (GDAL >= 1.11) YES if RGB must be resampled to YCbCr 4:2:0. Defaults to NO.
- **YCC=YES/NO** : (GDAL >= 2.0) YES if RGB must be transformed to YCC color space (“MCT transform”, i.e. internal transform, without visual degradation). Defaults to YES.
- **NBITS=int_value** : (GDAL >= 2.0) Bits (precision) for sub-byte files (1-7), sub-uint16 (9-15), sub-uint32 (17-31).
- **1BIT_ALPHA=YES/NO**: (GDAL >= 2.0) 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**: (GDAL >= 2.0) Whether to force encoding last channel as alpha channel. Only useful if the color interpretation of that channel is not already Alpha. Defaults to NO.
- **PROFILE=AUTO/UNRESTRICTED/PROFILE_1**: (GDAL >= 2.0) 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 advertize it in the relevant case. Note that the default values of those options are compatible with PROFILE_1. Otherwise UNRESTRICTED is advertized. Defaults to AUTO.

- **INSPIRE_TG=YES/NO:** (GDAL >= 2.0) Whether to use JPEG2000 features that comply with [Inspire Orthoimagery Technical Guidelines](#). Defaults to NO. 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:** (GDAL >= 2.0) Whether to advertize JPX features, and add a Reader requirement box, when a GMLJP2 box is written. Defaults to YES. This option should not be used unless compatibility problems with a reader occur.
- **GEOBOXES_AFTER_JP2C=YES/NO:** (GDAL >= 2.0) 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},...:** (GDAL >= 2.0) 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:** (GDAL >= 2.0) Whether to generate tile-parts and according to which criterion. Defaults to DISABLED.
- **CODEBLOCK_WIDTH=int_value:** (GDAL >= 2.0) Codeblock width: power of two value between 4 and 1024. Defaults to 64. 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=int_value:** (GDAL >= 2.0) Codeblock height: power of two value between 4 and 1024. Defaults to 64. 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=string:** (GDAL >= 2.4 and 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.

- **WRITE_METADATA=YES/NO:** (GDAL >= 2.0) 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.

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:** (GDAL >= 2.0) (Only if WRITE_METADATA=YES) Whether only metadata from the main domain should be written. Defaults to NO.
- **USE_SRC_CODESTREAM=YES/NO:** (GDAL >= 2.0) (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).

4.84.5.1 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)

4.84.5.2 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": {
```

(continues on next page)

(continued from previous page)

```

    "#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": [
        "New in GDAL 2.2",
        "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": [
        "New in GDAL 2.2",
        "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": [
        "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",

```

(continues on next page)

(continued from previous page)

```

        "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",

    {
        "#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)",

```

(continues on next page)

(continued from previous page)

```

        "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.",
        "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](#).

4.84.6 Vector information

Starting with GDAL 2.0, 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.

4.84.7 See Also

- Implemented as `gdal/frmts/openjpeg/openjpegdataset.cpp`.
- [Official JPEG-2000 page](#)
- [The OpenJPEG library home page](#)
- [OGC GML in JPEG2000, version 2.0 \(GDAL 2.0 and 2.1\)](#)
- [OGC GML in JPEG2000, version 2.0.1 \(GDAL 2.2 and above\)](#)
- [Inspire Data Specification on Orthoimagery - Technical Guidelines](#)

Other JPEG2000 GDAL drivers :

- *JP2000: based on Jasper library (open source)*
- *JP2ECW: based on Erdas ECW library (proprietary)*
- *JP2MRSID: based on LizardTech MrSID library (proprietary)*
- *JP2KAK: based on Kakadu library (proprietary)*

4.85 JPEG2000 – Implementation of the JPEG-2000 part 1

Driver short name

JPEG2000

Build dependencies

libjasper

This implementation based on JasPer software (see below).

XMP metadata can be extracted from JPEG2000 files, and will be stored as XML raw content in the xml:XMP metadata domain.

4.85.1 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.*)

4.85.2 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 most priority): PAM, GeoJP2, GMLJP2, WORLDFILE.

Starting with GDAL 2.2, 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. First mentioned sources are the most priority over the next 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.

4.85.3 Option Options

(GDAL >= 2.0) The following open option is available:

- **1BIT_ALPHA_PROMOTION=YES/NO**: Whether a 1-bit alpha channel should be promoted to 8-bit. Defaults to YES.
- **GEOREF_SOURCES=string**: (GDAL > 2.2) Define which georeferencing sources are allowed and their priority order. See [Georeferencing](#) paragraph.

4.85.4 Creation Options

- **WORLDFILE=ON**: Force the generation of an associated ESRI world file (.wld).
- **FORMAT=JP2|JPC**: Specify output file format.
- **GMLJP2=YES/NO**: (Starting with GDAL 2.0) 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. As implemented currently, the GMLJP2 box will be written after the codestream. Defaults to YES.
- **GMLJP2V2_DEF=filename**: (Starting with GDAL 2.0) 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. As implemented currently, the GMLJP2 box will be written after the codestream.
- **GeoJP2=YES/NO**: (Option starting with GDAL 2.0, but already enabled in previous versions. Require a modified Jasper with GeoJP2 support enabled) Indicates whether a UUID/GeoTIFF box conforming to the GeoJP2 (GeoTIFF in JPEG2000) specification should be included in the file. Defaults to YES.
- **NBITS=int_value** : (GDAL >= 2.3) Bits (precision) for sub-byte files (1-7), sub-uint16 (9-15).
- Encoding parameters, directly delivered to the Jasper library described in the Jasper documentation. Quoted from the docs:

``The following options are supported by the encoder:

imagearea-tl-x=x	Set the x-coordinate of the top-left corner of the image area to x.	
imagearea-tl-y=y	Set the y-coordinate of the top-left corner of the image area to y.	
tile-grid-tl-x=x	Set the x-coordinate of the top-left corner of the tiling grid to x.	
tile-grid-tl-y=y	Set the y-coordinate of the top-left corner of the tiling grid to y.	
tile-width=w	Set the nominal tile width to w.	
tile-height=h	Set the nominal tile height to h.	
precinct-width=w	Set the precinct width to w. The argument w must be an integer power of two. The default value is 32768.	
precinct-height=h	Set the precinct height to h. The argument h must be an integer power of two. The default value is 32768.	
code-block-width=w	Set the nominal code block width to w. The argument w must be an integer power of two. The default value is 64.	
code-block-height=h	Set the nominal code block height to h. The argument h must be an integer power of two. The default value is 64.	
mode=m	Set the coding mode to m. The argument m must have one of the following values: ===== Value Description ===== int integer mode real real mode ===== If lossless coding is desired, the integer mode must be used. By default, the integer mode is employed. The choice of mode also determines which multicomponent and wavelet transforms (if any) are employed.	
rate=r	Specify the target rate. The argument r is a positive real number. Since a rate of one corresponds to no compression, one should never need to explicitly specify a rate greater than one. By default, the target rate is considered to be infinite.	
layer-rates=rate1,rate2,...,rateN	Specify the rates for any intermediate layers. The argument to this option is a comma separated list of N rates. Each rate is a positive real number. The rates must increase monotonically. The last rate in the list should be less than or equal to the overall rate (as specified with the rate option).	
progression=p	Set the progression order to p. The argument p must have one of the following values: ===== Value Description ===== lrcp layer-resolution-component-position (LRCP) progressive (i.e., rate scalable) rlcp resolution-layer-resolution-component-position (RLCP) progressive (i.e., resolution scalable) rpcl resolution-position-component-layer (RPCL) progressive pcrl position-component-resolution-layer (PCRL) progressive cpcl component-position-resolution-layer (CPRL) progressive =====	
no-multicomponent	Disallow the use of any multicomponent transform.	
number-resolution-levels=n	Set the number of resolution levels to n. The argument n must be an integer that is greater than or equal to one. The default value is 6.	
sop	Generate SOP marker segments.	
eph	Generate EPH marker segments.	
lazy	Enable lazy coding mode (a.k.a. arithmetic coding bypass).	
terminate	Terminate all coding passes.	
segmentation	Use segmentation symbols.	
vertical-causal-contexts	Use vertically stripe causal contexts.	
predictable-termination	Use predictable termination.	
reset-probability-models	Reset the probability models after each coding pass.	

4.85.5 See Also

- Implemented as `gdal/frmts/jpeg2000/jpeg2000dataset.cpp`.
- You need modified JasPer library to build this driver with GeoJP2 support enabled. Modified version can be downloaded from <http://download.osgeo.org/gdal/jasper-1.900.1.uuid.tar.gz>
- [Official JPEG-2000 page](#)
- [The JasPer Project 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)*

4.86 JPEGLS

Driver short name

JPEGLS

Build dependencies

CharLS library

This driver is an implementation of a JPEG-LS reader/writer based on the Open Source CharLS library (BSD style license).

The driver can read and write lossless or near-lossless images. Note that it is not aimed at dealing with too big images (unless enough virtual memory is available), since the whole image must be compressed/decompressed in a single operation.

4.86.1 Driver capabilities

Supports `CreateCopy()`

This driver supports the `GDALDriver::CreateCopy()` operation

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.86.2 Creation Options

- **INTERLEAVE=PIXEL/LINE/BAND** : Data interleaving in compressed stream. Default to BAND.
- **LOSS_FACTOR=error_threshold** : 0 (the default) means loss-less compression. Any higher value will be the maximum bound for the error.

4.86.3 See Also:

- Implemented as `gdal/frmts/jpegls/jpeglsdataset.cpp`.
- [Homepage of the CharLS library](#)

4.87 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.

Starting with GDAL 1.10.1, 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` to `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.

To be able to read and write JPEG images with 12-bit sample, you can build GDAL with its internal libjpeg (based on IJG libjpeg-6b, with additional changes for 12-bit sample support), or explicitly pass `-with-jpeg12=yes` to configure script when building with external libjpeg. See “[8 and 12 bit JPEG in TIFF](#)” wiki page for more details.

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 1.9.0, XMP metadata can be extracted from the file, and will be stored as XML raw content in the xml:XMP metadata domain.

Starting with GDAL 2.0, embedded EXIF thumbnails (with JPEG compression) can be used as overviews, and generated by GDAL.

4.87.1 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.)*

4.87.2 Color Profile Metadata

Starting with GDAL 1.11, 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.

4.87.3 Error management

While decoding, libjpeg has resiliency towards some errors in the JPEG datastream and will try to recover from them as much of possible. Starting with GDAL 1.11.2, 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.

4.87.4 Creation Options

JPEG files are created using the “JPEG” driver code. Only Byte band types are supported, and only 1 and 3 band (RGB) configurations. JPEG file creation is implemented by the batch (CreateCopy) method. YCbCr, CMYK or YCbCrK colorspace are 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.

- **WORLDFILE=YES**: Force the generation of an associated ESRI world file (with the extension .wld).
- **QUALITY=n**: 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 10-100. Low values result in higher compression ratios, but poorer image quality. Values above 95 are not meaningfully better quality but can be substantially larger.

- **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. Starting with GDAL 1.10, this can be disabled by setting this option to NO.
- **ARITHMETIC=YES/NO**: (Starting with GDAL 1.10) To enable arithmetic coding. Not enabled in all libjpeg builds, because of possible legal restrictions.
- **BLOCK=1...16**: (Starting with GDAL 1.10 and 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 or RGB1**: (Starting with GDAL 1.10 and libjpeg 9). Set to RGB1 for lossless RGB. Note: this will produce files incompatible with versions prior to libjpeg 9.
- **SOURCE_ICC_PROFILE=value**: (Starting with GDAL 1.11). ICC profile encoded in Base64.
- **COMMENT=string**: (Starting with GDAL 2.0). String to embed in a comment JPEG marker. When reading, such strings are exposed in the COMMENT metadata item.
- **EXIF_THUMBNAIL=YES/NO**: (Starting with GDAL 2.0). Whether to generate an EXIF thumbnail(overview), itself JPEG compressed. Defaults to NO. If enabled, the maximum dimension of the thumbnail will be 128, if neither THUMBNAIL_WIDTH nor THUMBNAIL_HEIGHT are specified.
- **THUMBNAIL_WIDTH=n**: (Starting with GDAL 2.0). Width of thumbnail. Only taken into account if EXIF_THUMBNAIL=YES.
- **THUMBNAIL_HEIGHT=n**: (Starting with GDAL 2.0). Height of thumbnail. Only taken into account if EXIF_THUMBNAIL=YES.
- **WRITE_EXIF_METADATA=YES/NO**: (Starting with GDAL 2.3). Whether to write EXIF_xxxx metadata items in a EXIF segment. Default to YES.

4.87.5 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

Continued on next page

Table 3 – continued from previous page

Metadata item name	Hex code	Type	Number of values	Optionality
EXIF_ComponentsConfiguration	0x9101	UNDEFINED	4	Mandatory
EXIF_CompressedBitsPerPixel	0x9102	RATIONAL	1	Optional
EXIF_ShutterSpeedValue	0x9201	SRATIONAL	1	Optional
EXIF_ApertureValue	0x9202	RATIONAL	1	Optional
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

Continued on next page

Table 3 – continued from previous page

Metadata item name	Hex code	Type	Number of values	Optionality
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

4.87.6 See Also

- Independent JPEG Group
- libjpeg-turbo
- *GTiff – GeoTIFF File Format*
- EXIF v2.31 specification

4.88 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.

4.88.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

4.88.2 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.



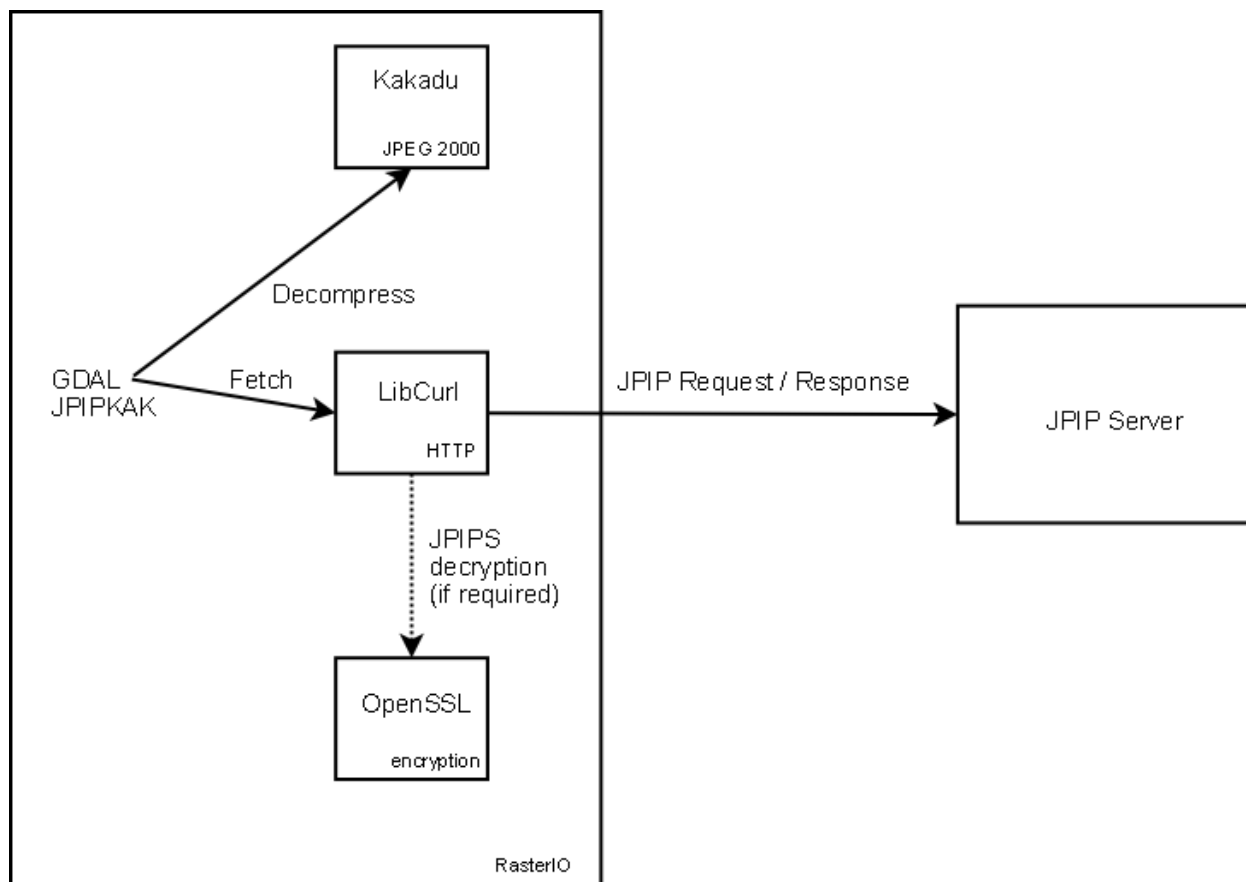
1. Initial JPIP request for a target image, a target id, a session over http, data to be returned as a jpp-stream are requested and a maximum length is put on the response. In this case no initial window is requested, though it can be. Server responds with a target identifier that can be used to identify 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 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.

4.88.3 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.



4.88.4 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).

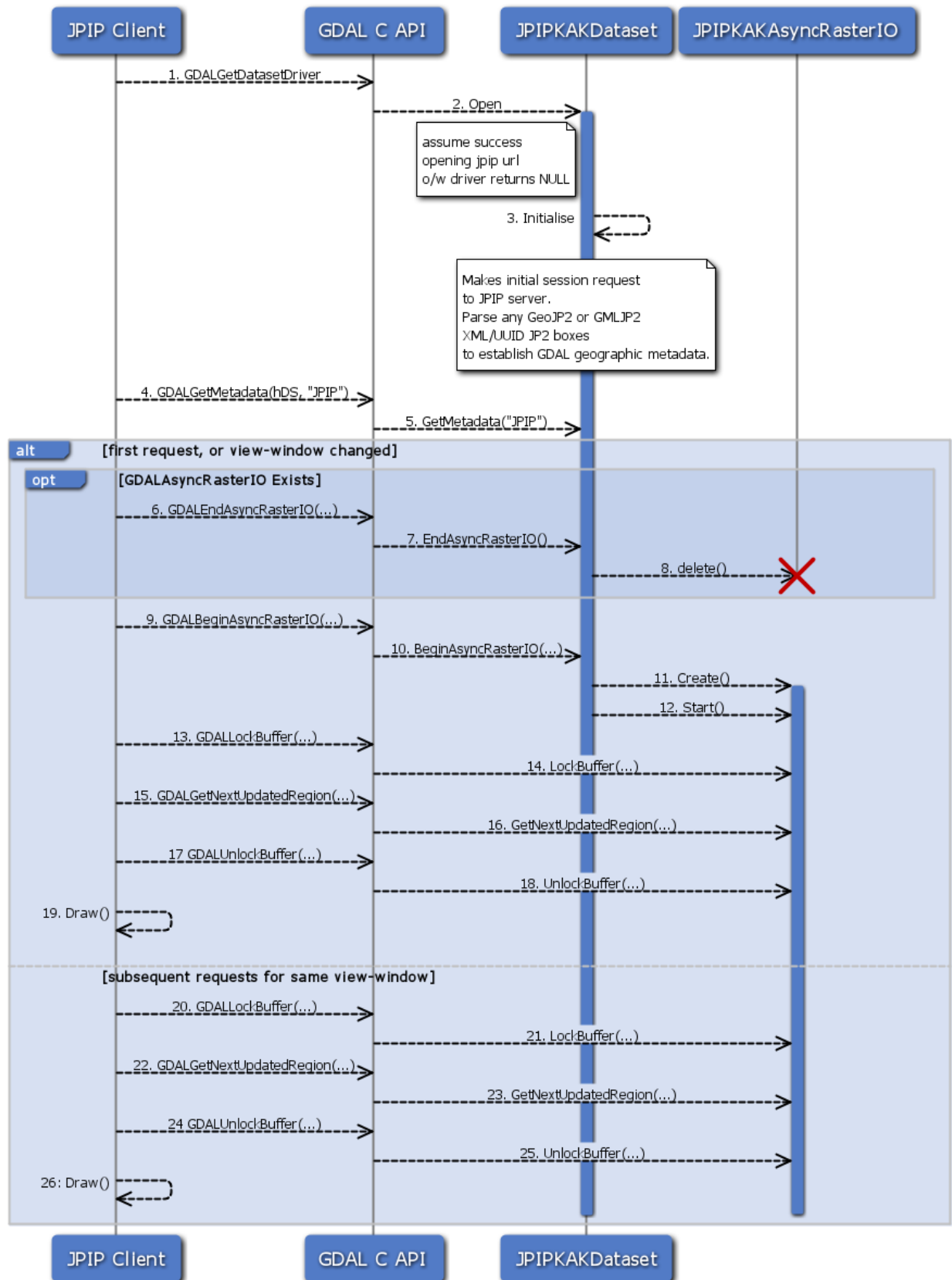
The driver uses a simple threading model to support requesting reads of the data and remote fetching. This threading model supports two separate client windows, with just one connection to the server. Requests to the server are multiplexed to utilize available bandwidth efficiently. The client identifies 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 `jpeg://`. 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_SPRECISSION”

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 JPIPKAAsyncRasterIO, a lock is acquired in JPIPKAAsyncRasterIO->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 JPIPKAAsyncDataset->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 JPIPKAAsyncRasterIO, a lock is acquired in JPIPKAAsyncRasterIO->GetNextUpdatedRegion

19. Draw

Client renders image data

20. *GDALLockBuffer*21. *LockBuffer*22. *GDALGetNextUpdatedRegion*23. *GetNextUpdatedRegion*24. *GDALUnlockBuffer*25. *UnlockBuffer*26. *Draw*

4.88.5 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.

4.88.6 See Also

- [JPEG 2000 Interactive Protocol \(Part 9 – JPIP\)](#)
- <http://www.opengeospatial.org/standards/gmljp2>
- [Kakadu Software](#)
- [IAS demo \(example JPIP\(S\) streams\)](#)

4.88.7 NOTES

Driver originally developed by [ITT VIS](#) and donated to GDAL to enable SSL enabled JPIP client streaming of remote JPEG 2000 datasets.

4.89 KEA

Driver short name

KEA

Build dependencies

libkea and libhdf5 libraries

Starting with GDAL 2.0, 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

4.89.1 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

New in version 3.0:

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.89.2 Creation options

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_value: The size of each block for image data. Defaults to 256
- **ATTBLOCKSIZE**=integer_value: The size of each block for attribute data. Defaults to 1000
- **MDC_NELMTS**=integer_value: Number of elements in the meta data cache. Defaults to 0. See the [Data caching](#) page of HDF5 documentation.
- **RDCC_NELMTS**=integer_value: Number of elements in the raw data chunk cache. Defaults to 512. See the [Data caching](#) page of HDF5 documentation.
- **RDCC_NBYTES**=integer_value: Total size of the raw data chunk cache, in bytes. Defaults to 1048576. See the [Data caching](#) page of HDF5 documentation.
- **RDCC_W0**=floating_point_value between 0 and 1: Preemption policy. Defaults to 0.75. See the [Data caching](#) page of HDF5 documentation.
- **SIEVE_BUF**=integer_value: Sets the maximum size of the data sieve buffer. Defaults to 65536. See [H5Pset_sieve_buf_size\(\)](#) documentation

- **META_BLOCKSIZE**=integer_value: Sets the minimum size of metadata block allocations. Defaults to 2048. See [H5Pset_meta_block_size\(\)](#) documentation
- **DEFLATE**=integer_value: Compression level between 0 (no compression) to 9 (max compression). Defaults to 1
- **THEMATIC**=YES/NO: If YES then all bands are set to thematic. Defaults to NO

4.89.3 See Also

- [libkea bitbucket repository](#)
- [The KEAimage file format](#), by Peter Bunting and Sam Gillingham, published in Computers&Geosciences
- [HDF5 driver page](#)

4.90 KMLSuperoverlay – KMLSuperoverlay

Driver short name

KMLSuperoverlay

Driver built-in by default

This driver is built-in by default

NOTE: Implemented as `gdal/frmts/kmlsuperoverlay/kmlsuperoverlay.cpp`.

4.90.1 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.*)

4.91 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 `gdal/frmts/raw/krodataset.cpp`.

4.91.1 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.)*

4.92 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 and writing Erdas 7.x .LAN and .GIS raster files. Currently 4bit, 8bit and 16bit pixel data types are supported for reading and 8bit and 16bit for writing.

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 `gdal/frmts/raw/landataset.cpp`

Development of this driver was financially supported by Kevin Flanders of (PeopleGIS).

4.92.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

4.93 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.

4.93.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

4.93.2 Georeference

Note, that GDAL simple affine georeference model 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.

Before GDAL 1.10.2, a maximum of 11 x 20 GCPs were reported. This might be unsuitable for correct warping. Starting with GDAL 1.10.2, a much higher density 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.

Starting with GDAL 1.11, 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.

4.93.3 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.

4.93.4 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.

Starting with GDAL 1.11, 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.

4.93.5 Subdatasets

NOAA <=14 datasets advertize 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/FAC data).

NOAA >=15 datasets advertize 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/FAC data).

NOAA >=15 datasets advertize 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.

4.93.6 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.

4.93.7 See Also

- Implemented as `gdal/frmts/l1b/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](#)

4.94 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 FARSITE, FLAMMAP, and FBAT (www.fire.org). The bands of an LCP file store data describing terrain, tree canopy, and surface fuel. The [LANDFIRE Data Distribution Site](http://www.landfire.gov) distributes data in LCP format, and programs such as FARSITE and [LFDAT](#) can create LCP files from a set of input rasters.

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 reads the linear unit, cell size, and extent, but the LCP file does not specify the projection. UTM projections are typical, but other projections are possible.

4.94.1 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.*)

4.94.2 Metadata

The GDAL LCP driver reports dataset- and band-level metadata:

4.94.2.1 Dataset

LATITUDE: Latitude of the dataset, negative for southern hemisphere

LINEAR_UNIT: Feet or meters

DESCRIPTION: LCP file description

4.94.2.2 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 gdal/frmts/raw. It should be implemented as gdal/frmts/raw/lcpdataset.cpp.

4.94.3 Creation Options

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]: Vertical unit for elevation band.

SLOPE_UNIT=[DEGREES/PERCENT]

ASPECT_UNIT=[AZIMUTH_DEGREES/GRASS_CATEGORIES/GRASS_DEGREES]

FUEL_MODEL_OPTION=[NO_CUSTOM_AND_NO_FILE/CUSTOM_AND_NO_FILE/NO_CUSTOM_AND_FILE/CUSTOM_AND_FILE]: Specify whether or not custom fuel models are used, and if a custom fuel model file is present.

CANOPY_COV_UNIT=[PERCENT/CATEGORIES]

CANOPY_HT_UNIT=[METERS_X_10/FEET/METERS/FEET_X_10]

CBH_UNIT=[METERS_X_10/METERS/FEET/FEET_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]

DUFF_UNIT=[MG_PER_HECTARE_X_10/TONS_PER_ACRE_X_10]

CALCULATE_STATS=[YES/NO]: 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]: 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]: 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=[...]: 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,
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
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

Table 5 – continued from

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 = custom
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 = lb/ft ³ x 100
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

4.95 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 9 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.

4.95.1 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.*)

4.95.2 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.

4.95.3 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`.

4.95.4 See Also:

- Implemented as `gdal/frmts/leveller/levellerdataset.cpp`.
- Visit [Daylon Graphics](#) for the Leveller SDK, which documents the Leveller format.

4.96 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 `gdal/frmts/raw/loaslasdataset.cpp`.

4.96.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*vsimem*/, etc.)

4.97 MAP – OziExplorer .MAP

Driver short name

MAP

Driver built-in by default

This driver is built-in by default

OziExplorer MAP files.

4.97.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.97.2 See Also:

- Implemented as `gdal/frmts/map/mapdataset.cpp`.
- [OziExplorer Map File Format](#)

4.98 MRF – Meta Raster Format

Driver short name

MRF

New in version 2.1.

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`”

4.98.1 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.)*

4.98.2 Links

- [MRF User Guide](#)
- [MRF Specification](#)
- [Source repository nasa-gibs/mrf](#)

4.99 MBTiles

Driver short name

MBTiles

Build dependencies

libsqlite3

The MBTiles driver allows reading rasters in the MBTiles format, which is a specification for storing tiled map data in SQLite databases.

Starting with GDAL 2.1, the MBTiles driver has creation and write support for MBTiles raster datasets.

Starting with GDAL 2.3, 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.

Starting with GDAL 2.3, 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 behaviour by defining the MBTILES_BAND_COUNT configuration option (or starting with GDAL 2.1, 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.

4.99.1 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.)*

4.99.2 Opening options

Starting with GDAL 2.1, 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: Whether to use the ‘bounds’ metadata, when available, to determine the AOI. Defaults to YES.
 - **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: 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. Defaults to AUTO.
 - **TILE_FORMAT**=PNG/PNG8/JPEG: Format used to store tiles. See *Tile formats* section. Only used in update mode. Defaults to PNG.
 - **QUALITY**=1-100: Quality setting for JPEG compression. Only used in update mode. Default to 75.
 - **ZLEVEL**=1-9: DEFLATE compression level for PNG tiles. Only used in update mode. Default to 6.
 - **DITHER**=YES/NO: Whether to use Floyd-Steinberg dithering (for **TILE_FORMAT**=PNG8). Only used in update mode. Defaults to NO.

- Vector only (GDAL >= 2.3):
 - **CLIP=YES/NO**: Whether to clip geometries of vector features to tile extent. Defaults to YES.
 - **ZOOM_LEVEL_AUTO=YES/NO**: Whether to auto-select the zoom level for vector layers according to the spatial filter extent. Only for display purpose. Defaults to NO.

4.99.3 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 (WebMercator) 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

4.99.3.1 Tile formats

MBTiles can store tiles in PNG or JPEG. Support for those tile formats depend if the underlying drivers are available in GDAL. By default, GDAL will PNG tiles.

It is possible to select the tile format by setting the creation/open option `TILE_FORMAT` to one of PNG, PNG8 or JPEG. 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.

4.99.4 Vector 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.

4.99.5 Creation options

The following creation options are available:

- Raster and vector:
 - **NAME**=string. Tileset name, used to set the ‘name’ metadata item. If not specified, the basename of the filename will be used.
 - **DESCRIPTION**=string. 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. The layer type, used to set the ‘type’ metadata item. Default to ‘overlay’.
- Raster only:
 - **VERSION**=string. The version of the tileset, as a plain number, used to set the ‘version’ metadata item. Default to ‘1.1’.
 - **BLOCKSIZE**=integer. (GDAL >= 2.3) Block/tile size in width and height in pixels. Defaults to 256. Maximum supported is 4096.
 - **TILE_FORMAT**=PNG/PNG8/JPEG: Format used to store tiles. See *Tile formats* section. Defaults to PNG.
 - **QUALITY**=1-100: Quality setting for JPEG compression. Default to 75.
 - **ZLEVEL**=1-9: DEFLATE compression level for PNG tiles. Default to 6.
 - **DITHER**=YES/NO: Whether to use Floyd-Steinberg dithering (for **TILE_FORMAT**=PNG8). Defaults to NO.
 - **ZOOM_LEVEL_STRATEGY**=AUTO/LOWER/UPPER. 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. Resampling algorithm. Defaults to BILINEAR.
 - **WRITE_BOUNDS**=YES/NO: Whether to write the bounds ‘metadata’ item. Defaults to YES.
- Vector only (GDAL >= 2.3):
 - **MINZOOM**=integer: Minimum zoom level at which tiles are generated. Defaults to 0.
 - **MAXZOOM**=integer: Minimum zoom level at which tiles are generated. Defaults to 5. Maximum supported value is 22
 - **CONF**=string: 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**=positive_integer. Number of units in a tile. The greater, the more accurate geometry coordinates (at the expense of tile byte size). Defaults to 4096
- **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. Whether to compress tiles with the Deflate/GZip algorithm. Defaults to YES. Should be left to YES for FORMAT=MBTILES.
- **TEMPORARY_DB**=string. 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. Maximum size of a tile in bytes (after compression). Defaults to 500 000. If a tile is greater than this threshold, features will be written with reduced precision, or discarded.
- **MAX_FEATURES**=integer. Maximum number of features per tile. Defaults to 200 000.
- **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.

4.99.6 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",
    "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 behaviour with the below layer creation options, although that is not convenient in the ogr2ogr use case.

4.99.7 Layer creation options (vector)

- **MINZOOM**=integer: Minimum zoom level at which tiles are generated. Defaults to the dataset creation option MINZOOM value.
- **MAXZOOM**=integer: Minimum zoom level at which tiles are generated. Defaults to the dataset creation option MAXZOOM value. Maximum supported value is 22
- **NAME**=string: 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**=string: A description of the layer.

4.99.8 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)

4.99.9 Vector tiles

Starting with GDAL 2.3, 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.

4.99.10 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"]],
```

(continues on next page)

(continued from previous page)

```

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 = (-3757031.2500000000000000,11271093.7500000000000000)
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
  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":
→"iVBORw0KGgoAAAANSUhEUgAAAGQAAABDEAIAAAC1uevOAAACXBIWXMAAABIAAAASABGyWs+AAAABmJLR0T/
→////////
→8JWPfcAAABPk1EQVR42u3cMRLBQBSA4Zc9CgqcALXC4bThBA5gNFyFM+wBVNFqjYTsZpfi1Sm++bOv2ETEdNK2pc/
→T9ny977rCn+fx8rjtc7dMmybnxXy9KncGWGCBBRZYYIEFFlhggQUWWGCBBRZYYIE1/
→GzSLB0CLLAUCyywWAILLLDAAGssGyFlcAqnJRiKRZYYIEFFlhggQUWWGDZCsFSLDDAAGssP4DazQowVIssMACy1ZYG6w
→HAYWWIplKwQLLLDAAGssZyywWAILLLDAqh6We4VgKZatECyWFAssMACCyywWAILLLBshWCBpVhggQUWWGCBBRZYYIFlKw
→fp8BhZYigUWWGB9C+t9ggUWWGD5FA44XxBz7mcwZM9VAAAAXJRFWHRkYXRlOmNyZWF0ZQAyMDE5LTA5LTAyVDIzOjI5OjI5
→"}</Json>

```

(continues on next page)

(continued from previous page)

```

    </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

```

4.99.11 See Also

- [MBTiles specification](#)
- [UTFGrid specification](#)
- *Mapbox Vector tiles driver*

4.100 MEM – In Memory Raster

Driver short name

MEM

Driver built-in by default

This driver is built-in by default

GDAL supports the ability to hold rasters 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.

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.

4.100.1 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
```

or

```
MEM:::DATAPOINTER=0x1467BEF0,PIXELS=100,LINES=100,BANDS=3,DATATYPE=Byte,
      PIXELOFFSET=3,LINEOFFSET=300,BANDOFFSET=1
```

- **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 `CPL-PrintPointer` 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.

4.100.2 Creation Options

There are no supported creation options.

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.

4.100.3 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

4.100.4 Multidimensional API support

New in version 3.1.

The MEM driver supports the *Multidimensional Raster Data Model*.

4.101 MFF – Vexcel MFF Raster

Driver short name

MFF

Driver built-in by default

This driver is built-in by default

GDAL includes read, update, and creation 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.

On creation (with a format code of MFF) a simple, ungeoreferenced raster file is created.

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 `gdal/frmts/raw/mffdataset.cpp`.

4.101.1 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.*)

4.102 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, update, and creation. 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.

Newly created files (with a type of MFF2) are always just raw rasters with no georeferencing information. For read, and creation all data types (real, integer and complex in bit depths of 8, 16, 32) should be supported.

IMPORTANT: When creating a new MFF2, be sure to set the projection before setting the geotransform (this is necessary because the geotransform is stored internally as 5 latitude-longitude ground control points, and the projection is needed to do the conversion).

NOTE: Implemented as `gdal/frmts/raw/hkvdataset.cpp`.

4.102.1 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

Supports VirtualIO

This driver supports *virtual I/O operations (vsimem/, etc.)*

4.102.2 Format Details

4.102.2.1 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.

4.102.2.2 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
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 }
```

4.102.2.3 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.

4.102.2.4 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         = 11
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)
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.

4.102.2.5 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.

4.102.2.6 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

4.102.2.7 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.

```

4.103 MG4Lidar – MrSID/MG4 LiDAR Compression / Point Cloud View files

Driver short name

MG4Lidar

Build dependencies

LIDAR SDK

This driver provides a way to view MrSID/MG4 compressed LiDAR file as a raster DEM. The specifics of the conversion depend on the desired cellsize, filter criteria, aggregation methods and possibly several other parameters. For this reason, **the best way to read a MrSID/MG4 compressed LiDAR file is by referencing it in a View (.view) file, which also parameterizes its raster-conversion. The driver will read an MG4 file directly, however it uses default rasterization parameters that may not produce a desirable output.** The contents of the View file are described in the specification [MrSID/MG4 LiDAR View Documents](#).

MrSID/MG4 is a wavelet-based point-cloud compression technology. You may think of it like a LAS file, only smaller and with a built in spatial index. It is developed and distributed by Extensis. This driver supports reading of MG4 LiDAR files using Extensis' decoding software development kit (DSDK). **This DSDK is freely distributed; but, it is not open source software. You should contact Extensis to obtain it (see link at end of this page).**

4.103.1 Example View files (from View Document specification)

4.103.1.1 Simplest possible .view file

The simplest way to view an MG4 file is to wrap it in a View (.view) file like this. Here, the relative reference to the MG4 file means that the file must exist in the same directory as the .view file. Since we're not mapping any bands explicitly, we get the default, which is elevation only. By default, we aggregate based on mean. That is, if two (or more) points land on a single cell, we will expose the average of the two. There's no filtering here so we'll get all the points regardless of classification code or return number. Since the native datatype of elevation is "Float64", that is the datatype of the band we will expose.

```
<PointCloudView>
  <InputFile>Tetons.sid</InputFile>
</PointCloudView>
```

4.103.1.2 Crop the data

This is similar to the example above but we are using the optional ClipBox tag to select a 300 meter North-South swatch through the cloud. If we wanted to crop in the East-West directions, we could have specified that explicitly instead of using NOFILTER for those. Similarly, we could also have cropped in the Z direction as well.

```
<PointCloudView>
  <InputFile>Tetons.sid</InputFile>
  <ClipBox>505500 505800 NOFILTER NOFILTER</ClipBox>
</PointCloudView>
```

4.103.1.3 Expose as a bare earth (Max) DEM

Here, we expose a single band (elevation) but we want only those points that have been classified as “Ground”. The ClassificationFilter specifies a value of 2 - the ASPRS Point Class code that stipulates “Ground” points. Additionally, instead of the default “Mean” aggregation method, we specify “Max”. This means that if two (or more) points land on a single cell, we expose the larger of the two elevation values.

```
<PointCloudView>
  <InputFile>E:\ESRIDevSummit2010\Tetons.sid</InputFile>
  <Band> <!-- Max Bare Earth-->
    <Channel>Z</Channel>
    <AggregationMethod>Max</AggregationMethod>
    <ClassificationFilter>2</ClassificationFilter>
  </Band>
</PointCloudView>
```

4.103.1.4 Intensity image

Here we expose an intensity image from the point cloud.

```
<PointCloudView>
  <InputFile>Tetons.sid</InputFile>
  <Band>
    <!-- All intensities -->
    <Channel>Intensity</Channel>
  </Band>
</PointCloudView>
```

4.103.1.5 RGB image

Some point cloud images include RGB data. If that’s the case, you can use a .view file like this to expose that data.

```
<PointCloudView>
  <InputFile>Grass Lake Small.xyzRGB.sid</InputFile>
  <Band>
    <Channel>Red</Channel>
  </Band>
  <Band>
    <Channel>Green</Channel>
  </Band>
  <Band>
    <Channel>Blue</Channel>
  </Band>
</PointCloudView>
```

4.103.2 Writing not supported

This driver does not support writing MG4 files.

4.103.3 Limitations of current implementation

Only one `<InputFile>` tag is supported. It must reference an MG4 file.

The only `<InterpolationMethod>` that is supported is `<None>` (default). Use this to specify a NODATA value if the default (maximum value of the datatype) is not what you want. See View Specification for details.

There is insufficient error checking for format errors and invalid parameters. Many invalid entries will likely fail silently.

4.103.4 See Also:

- Implemented as `gdal/frmts/mrsid_lidar/gdal_MG4Lidar.cpp`
- *MrSID/MG4 LiDAR View Document Specification*
- [Extensis web site](#)

4.103.4.1 Specification for MrSID/MG4 LiDAR View Documents

Version 1.0

Introduction

This document specifies the contents of an XML document used as a “view” into a LiDAR point cloud. It is intended to be a rigorous definition of an XML-based format for specifying rasterization of point cloud data. If you are looking for “something to get me started very quickly”, please see the examples below.

Document Structure (informative)

The overall element structure of a View document is informally shown below. Indentation and regular expression syntax are used to intuitively indicate parent-child nesting and the number of occurrences of elements.

```
PointCloudView
  InputFile +
  Datatype ?
  Band *
    Channel ?
    ClassificationFilter ?
    ReturnNumberFilter ?
    AggregationMethod ?
    InterpolationMethod ?
  ClassificationFilter ?
  ReturnNumberFilter ?
  AggregationMethod ?
  InterpolationMethod ?
  ClipBox ?
  CellSize ?
  GeoReference ?
```

Elements

Each element is specified as follows:

```

ElementName
  Cardinality:  number of occurrences allowed
  Parents:    what element(s) may contain this element
  Contents:   what may be placed inside the element
  Attributes: what attributes are allowed, if any
  Notes:     additional usage information or restriction

```

PointCloudView

Description: the root element for the document

Cardinality: 1

Parents: none (must be root element)

Contents: child elements as specified below:

- InputFile
- Datatype
- Band
- ClassificationFilter
- ReturnNumberFilter
- AggregationMethod
- InterpolationMethod
- ClipBox
- CellSize
- GeoReference

Attributes:

- version - this attribute must be present and set to the value 1.0

Notes: (none)

InputFile

Description: specifies an input file containing point cloud data

Cardinality: 1..n

Parents: PointCloudView

Contents: string (corresponding to a filename)

Attributes: (none)

Notes:

- Typically the file will be a MrSID/MG4 LiDAR file, but may also be a LAS file.

- The file name given may have a relative or absolute path. If relative, the path is to be expanded relative to the directory containing this View document.

Datatype

Description: specifies the datatype to which channel data should be coerced

Cardinality: 0 or 1

Parents: PointCloudView

Contents: string (corresponding to a datatype name)

Attributes: (none)

Notes:

- If this element is not present, the native datatype of the channel is used.
- Legal values are derived from those returned by GDALGetDataTypeByName, as follows:
 - Byte
 - UInt16
 - Int16
 - UInt32
 - Int32
 - Float32
 - Float64
- Channel data will be coerced via a c-style cast, truncating data as necessary.

Band

Description: list of which band(s) to expose and in what manner to process the band data

Cardinality: 0, 1 or 3

Parents: PointCloudView

Contents: child elements as follows:

- 0 or 1 Channel element
- 0 or 1 ClassificationFilter element
- 0 or 1 ReturnNumberFilter element
- 0 or 1 InterpolationMethod element
- 0 or 1 AggregationMethod element

Attributes: (none)

Notes:

- Not specifying any bands is the same as specifying only one with all default values.

Channel

Description: the name of the channel in the input file

Cardinality: 0 or 1 per Band element

Parents: Band

Contents: we use the following canonical names of channels

- X
- Y
- Z
- Intensity
- ReturnNum
- NumReturns
- ScanDir
- EdgeFlightLine
- ClassId
- ScanAngle
- UserData
- SourceId
- GPSTime
- Red
- Green
- Blue

Attributes: (none)

Notes:

- Custom channels have non-canonical names, are supported, and may be specified.
- If this element is omitted, the Channel for the Band shall default to Z.
- The channel names are derived from PointData.h of the MG4 Decode SDK.

ClassificationFilter

Description: A filter for points whose classification code is one of the specified values.

Cardinality: 0 or 1 per Band element

Parents: Band or PointCloudView

Contents: space-separated “Classification Values” (0-31) as defined by ASPRS Standard LIDAR Point Classes in the LAS 1.3 Specification.

Attributes: (none)

Notes:

- If this element is omitted, the band shall have no classification filter applied.

- If this element is a child of the PointCloudView element, it applies to all bands (unless overridden for a specific band)
- If this element is a child of a Band element, it applies to this band only and overrides any other setting
- Note that numbers are used to represent the filters, rather than strings. This is because there is no canonical, simple naming convention for them, and is also in keeping with existing practice in certain existing applications.

ReturnNumberFilter

Description: A filter for points whose return number is one of the specified values.

Cardinality: 0 or 1 per Band element

Parents: Band or PointCloudView

Contents: space-separated numbers (1, 2, ...) or the string LAST

Attributes: (none)

Notes:

- If this element is omitted, the band shall have no return number filter applied
- If this element is a child of the PointCloudView element, it applies to all bands (unless overridden for a specific band)
- If this element is a child of a Band element, it applies to this band only and overrides any other setting

AggregationMethod

Description: Each cell (pixel) can expose a single value. When 2 or more points fall on a single cell, this method determines what value to expose.

Cardinality: 0 or 1 per Band element

Parents: Band or PointCloudView

Contents: a string, one of Min, Max, or Mean

Attributes: (none)

Notes:

- If this element is omitted, the band shall have the “Mean” aggregation method applied
- If this element is a child of the PointCloudView element, it applies to all bands (unless overridden for a specific band)
- If this element is a child of a Band element, it applies to this band only and overrides any other setting

InterpolationMethod

Description: Method and parameter to interpolate NODATA values. Also specifies what the NODATA value is.

Cardinality: 0 or 1 per Band element

Parents: Band or PointCloudView

Contents: exactly one of the following elements:

- None
- InverseDistanceToAPower
- MovingAverage
- NearestNeighbor
- Minimum
- Maximum
- Range

Attributes: (none)

Notes

- Each of the interpolation methods (MovingAverage, etc.) is an element whose content is a text string corresponding to the parameter(s) for that method. See *GDAL Grid Tutorial* for a description of the methods and their parameter strings.
- In the parameter descriptions, MAX is used to indicate the value defined by libc which is the largest supportable value for the output datatype. If you choose to override this default be sure that the number you specify will fit in the datatype you specify.
- If this element is omitted, the band shall have the “None” interpolation method applied.
- If this element is a child of the PointCloudView element, it applies to all bands (unless overridden for a specific band)
- If this element is a child of a Band element, it applies to this band only and overrides any other setting

ClipBox

Description: geographic extent of region to be viewed

Cardinality: 0 or 1

Parents: PointCloudView

Contents: 4 or 6 doubles; the string NOFILTER may be specified in place of a double value

Attributes: (none)

Notes:

- The full 6 values are (in order): xmin, xmax, ymin, ymax, zmin, zmax.
- The string NOFILTER means to use the corresponding value of the Minimum Bounding Rectangle (MBR) of the input files. The point is not filtered by that value.
- If only 4 double are present, the zmin and zmax are assumed to be NOFILTER.
- If this element is not present, the clip box is assumed to be the MBR of the input files.

CellSize

Description: Side length of a (square) pixel in ground units

Cardinality: 0 or 1

Parents: PointCloudView

Contents: 1 double

Attributes: (none)

Notes:

- This element is used to determine the size of the resulting raster.
- If this element is omitted, the default cell size is the average (linear) point spacing (assuming a uniform distribution over the entire extent).

GeoReference

Description: the coordinate reference system of the view

Cardinality: 0 or 1

Parents: PointCloudView

Contents: a string (corresponding to a WKT)

Attributes: (none)

Notes:

- If this element is omitted, the WKT of the input files is used. If two or more files have different WKTs, then no GeoReference is defined.
- A typical use of this element is for when the MG4 file was created without adequate GeoReference information: cases where some combination of UOM, HorizCS and VertCS are missing are quite common.

Additional Requirements

Any element not recognized should be treated as an error.

Any attribute not recognized should be treated as an error.

This specification does not mandate the lexical ordering of the child elements within a given parent.

Examples

Simplest possible .view file

The simplest way to view an MG4 file is to wrap it in a View (.view) file like this. Here, the relative reference to the MG4 file means that the file must exist in the same directory as the .view file. Since we're not mapping any bands explicitly, we get the default, which is elevation only. By default, we aggregate based on mean. That is, if two (or more) points land on a single cell, we will expose the average of the two. There's no filtering here so we'll get all the points regardless of classification code or return number. Since the native datatype of elevation is "Float64", that is the datatype of the band we will expose.

```
<PointCloudView>
  <InputFile>Tetons.sid</InputFile>
</PointCloudView>
```

Crop the data

This is similar to the example above but we are using the optional ClipBox tag to select a 300 meter North-South swatch through the cloud. If we wanted to crop in the East-West directions, we could have specified that explicitly instead of using NOFILTER for those. Similarly, we could also have cropped in the Z direction as well.

```
<PointCloudView>
<InputFile>Tetons.sid</InputFile>
<ClipBox>505500 505800 NOFILTER NOFILTER</ClipBox>
</PointCloudView>
```

Expose as a bare earth (Max) DEM

Here, we expose a single band (elevation) but we want only those points that have been classified as “Ground.” The ClassificationFilter specifies a value of 2 - the ASPRS Point Class code that stipulates “Ground” points. Additionally, instead of the default “Mean” aggregation method, we specify “Max.” This means that if two (or more) points land on a single cell, we expose the larger of the two elevation values.

```
<PointCloudView>
  <InputFile>E:\ESRIDEvSummit2010\Tetons.sid</InputFile>
  <Band> <!-- Max Bare Earth-->
    <Channel>Z</Channel>
    <AggregationMethod>Max</AggregationMethod>
    <ClassificationFilter>2</ClassificationFilter>
  </Band>
</PointCloudView>
```

Intensity image

Here we expose an intensity image from the point cloud.

```
<PointCloudView>
  <InputFile>Tetons.sid</InputFile>
  <Band>
    <!-- All intensities -->
    <Channel>Intensity</Channel>
  </Band>
</PointCloudView>
```

RGB image

Some point cloud images include RGB data. If that's the case, you can use a .view file like this to expose that data.

```
<PointCloudView>
  <InputFile>Grass Lake Small.xyzRGB.sid</InputFile>
  <Band>
    <Channel>Red</Channel>
  </Band>
  <Band>
    <Channel>Green</Channel>
  </Band>
  <Band>
    <Channel>Blue</Channel>
  </Band>
</PointCloudView>
```

4.104 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.

4.104.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

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.

4.104.3 Georeference

4.104.4 See Also:

- Implemented as `gdal/frmts/mrsid/mrsiddataset.cpp`.
- [Extensis web site](#)

4.105 MSG – Meteosat Second Generation

MSG

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.

4.105.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

4.105.2 Build Instructions

Clone the EUMETSAT library for wavelet decomposition into `frmts/msg`.

If you are building with Visual Studio 6.0, extract the `.vc` makefiles for the `PublicDecompWT` from the file *PublicDecompWTMakefiles.zip* stored in that directory.

If you build using the GNU Makefile, use `-with-msg` option to enable MSG driver:

```
./configure --with-msg
```

If you find that some adjustments are needed in the makefile and/or the `msg` source files, please “commit” them. The EUMETSAT library promises to be “platform independent”, but as we are working with Microsoft Windows and Visual Studio 6.0, we did not have the facilities to check if the rest of the `msg` driver is. Furthermore, apply steps 4 to 7 from the *Raster driver implementation tutorial*, section “Adding Driver to GDAL Tree”.

MSG Wiki page is available at <http://trac.osgeo.org/gdal/wiki/MSG>. It’s dedicated to document building and usage hints

4.105.3 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 $\text{mW/m}^2/\text{sr}/(\text{cm}^{-1})^{-1}$. The result is Float32.
 - * L to perform radiometric calibration and get the result in $\text{W/m}^2/\text{sr}/\mu\text{m}$. 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 $\text{mW/m}^2/\text{sr}/(\text{cm}^{-1})^{-1}$:

```
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
```

4.105.4 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.

4.105.5 See Also

- Implemented as `gdal/frmts/msg/msgdataset.cpp`.
- <http://www.eumetsat.int> - European Organisation for the Exploitation of Meteorological Satellites

4.106 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.

4.106.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.107 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 `gdal/frmts/raw/ndfdataset.cpp`.

See Also: [NLAPS Data Format Specification](#).

4.107.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.108 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 `gdal/frmts/netcdf/netcdfdataset.cpp`.

4.108.1 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

4.108.2 Multiple Image Handling (Subdatasets)

Network Command 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
  NC_GLOBAL#table_id=Table 01 (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 sub-dataset, 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)
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.

4.108.3 Dimension

The NetCDF driver assume that data follows the CF-1 convention from [UNIDATA](#). The dimensions inside the NetCDF file use the following rules: (Z,Y,X). If there are more than 3 dimensions, the driver will merge them into bands. For example if you have an 4 dimension arrays of the type (P, T, Y, X). The driver will multiply the last 2 dimensions (P*T). The driver will display the bands in the following order. It will first increment T and then P. Metadata will be displayed on each band with its corresponding T and P values.

4.108.4 Georeference

There is no universal way of storing georeferencing in NetCDF files. The driver first tries to follow the CF-1 Convention from UNIDATA looking for the Metadata named “grid_mapping”. If “grid_mapping” is not present, the driver will try to find an lat/lon grid array to set geotransform array. The NetCDF driver verifies that the Lat/Lon array is equally space.

If those 2 methods fail, NetCDF driver will try to read the following metadata directly and set up georeferencing.

- spatial_ref (Well Known Text)
- GeoTransform (GeoTransform array)

or,

- Northernmost_Northing
- Southernmost_Northing
- Easternmost_Easting
- Westernmost_Easting

4.108.5 Open options

The following open options are available:

- **HONOUR_VALID_RANGE=YES/NO:** (GDAL > 2.2) Whether to set to nodata pixel values outside of the validity range indicated by valid_min, valid_max or valid_range attributes. Default is YES.

4.108.6 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.

4.108.7 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
```

4.108.8 Driver Improvements (GDAL >= 1.9.0)

The driver has undergone significant changes in GDAL 1.9.0, see NEWS file and [NetCDF Improvements](#).

4.108.8.1 Important Changes

- Added support for NC2, NC4 and NC4C file types for reading and writing, and HDF4 for reading. See [NetCDF File Format](#) for details.
 - NC : NetCDF Classic Format: The Original Binary Format.
 - NC2 : 64-bit Offset Format: Supporting Larger Variables
 - NC4 : NetCDF-4 Format: Uses HDF5
 - NC4C : NetCDF-4 Classic Model Format: HDF5 with NetCDF Limitations
 - HDF4 : HDF4 SD Format
- Improved support for CF-1.5 projected and geographic SRS reading and writing
- Improvements to metadata (global and variable) handling
- Added simple progress indicator
- Added support for DEFLATE compression (reading and writing) and szip (reading) - requires NetCDF-4 support
- Added support for valid_range/valid_min/valid_max
- Proper handling of signed/unsigned byte data
- Added support for Create() function - enables to use NetCDF directly with gdalwarp
- Added support for CF two-dimensional coordinate variables (see [CF Conventions](#)) via GDAL GEOLOCATION arrays (see rfc-4)

4.108.8.2 Creation Options

- **FORMAT=[NC/NC2/NC4/NC4C]:** Set the NetCDF file format to use, NC is the default. NC2 is normally supported by recent NetCDF installations, but NC4 and NC4C are available if NetCDF was compiled with NetCDF-4 (and HDF5) support.
- **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]:** 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]:** 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 Configuration Option 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.

- **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.

4.108.8.3 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 in GDAL trac).

4.108.9 VSI Virtual File System API support

Since GDAL 2.4, and with Linux kernel ≥ 4.3 and libnetcdf ≥ 4.5 , read operations on /vsi file systems are supported.

4.108.10 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 childs as specified in [Support of groups in CF](#)

4.108.11 Multidimensional API support

New in version 3.1.

The netCDF driver supports the *Multidimensional Raster Data Model* for reading and creation operations.

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.

4.108.12 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.

See [NetCDF GDAL wiki](#) for build instructions and information regarding HDF4, NetCDF-4 and HDF5.

4.108.13 See Also:

- *Vector side of the netCDF driver.*
- [NetCDF CF-1.5 convention](#)
- [NetCDF compiled libraries](#)
- [NetCDF Documentation](#)

4.109 NGS GEOID - 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.

4.109.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.109.2 See also

- [Description of NGS Geoid Height Grids format](#)
- [GEOID09 main page](#)
- [USGG2009 main page](#)

4.110 NGW – NextGIS Web

New in version 2.4.

Driver short name

NGW

Build dependencies

libcurl

NextGIS Web - is a server GIS, which allows to store and edit 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](#)).

4.110.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

4.110.2 Driver

NextGIS Web supports several raster types:

- Raster style
- Vector style
- WMS layer
- WMS Service
- Web map as combination of raster and vector styles

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 only raster and vector styles and WMS layers. You can get raster data as tiles or image (only tiles are supported now).

The driver supports read and copy from existing source dataset operations on rasters.

4.110.3 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.

4.110.4 Configuration options

The following configuration options are available:

- **NGW_USERPWD**: User name and password separated with colon. Optional and can be set using open options.
- **NGW_CACHE_EXPIRES**: 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).
- **NGW_CACHE_MAX_SIZE**: The cache maximum size in bytes. If cache reached maximum size, expired cached files will be deleted. Defaults to 67108864 (64Mb).
- **NGW_JSON_DEPTH**: The depth of json response that can be parsed. If depth is greater than this value, parse error occurs.

4.110.5 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.

4.110.6 Open options

The following open options are available:

- 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.

4.110.7 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.

4.110.8 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.

4.110.9 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
```

4.110.10 See also

- *Vector side of the driver*
- [NextGIS Web documentation](#)
- [NextGIS Web for developers](#)

4.111 NITF – National Imagery Transmission Format

Driver short name

NITF

Driver built-in by default

This driver is built-in by default

4.111.1 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.

4.111.1.1 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.

Since GDAL 1.8.0, 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.

4.111.1.2 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.

4.111.1.3 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          00020021216151629xxxxxxxxxxxxxxxxxxxxxxxxxxxx
DATA_0=This is test text file 01.

HEADER_1=TE          00020021216151629xxxxxxxxxxxxxxxxxxxxxxxxxxxx
DATA_1=This is test text file 02.

HEADER_2=TE          00020021216151629xxxxxxxxxxxxxxxxxxxxxxxxxxxx
DATA_2=This is test text file 03.

HEADER_3=TE          00020021216151629xxxxxxxxxxxxxxxxxxxxxxxxxxxx
DATA_3=This is test text file 04.

HEADER_4=TE          00020021216151629xxxxxxxxxxxxxxxxxxxxxxxxxxxx
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.

Since GDAL 1.8.0, 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.

Since GDAL 1.8.0, to add TEXT data to a NITF image, you can also pass creation options in the following format:

```
TEXT=HEADER_0=TE      00020021216151629xxxxxxxxxxxxxxxxxxxxxxxxxxxx
TEXT=DATA_0=This is test text file 01.
TEXT=HEADER_1=TE      00020021216151629xxxxxxxxxxxxxxxxxxxxxxxxxxxx
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.

4.111.1.4 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 0000000000000000
                  0000
```

(continues on next page)

(continued from previous page)

[illegible]

4.111.1.5 TREs as xml:TRE

Starting with GDAL 1.9.0, all TREs found in file and matching one of the TRE description of the `nifv_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="NPART" 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)

```
</tre>
</tres>
```

4.111.1.6 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 Tk1URjAyLjAwMDEgICAgVTIxN0cwSjA...
  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.

GDAL supports reading of several subtypes of NITF image files, and writing simple NITF 2.1 files. NITF 1.1, NITF 2.0, NITF 2.1 and NSIF 1.0 files with uncompressed, ARIDPCM, JPEG compressed, JPEG2000 (with Kakadu, ECW SDKs or other JPEG2000 capable driver) or VQ compressed images should be readable.

The read support test has been tested on various products, including CIB and CADRG frames from RPF products, ECRG frames, HRE products.

Color tables for pseudocolored images are read. In some cases nodata values may be identified.

Lat/Long extents are read from the IGEOLO 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 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 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 metadata. Starting with GDAL 2.2, 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.

4.111.2 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.*)

4.111.3 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 and ICORDS creation options are supplied.

Creation Options:

- 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.
- **IC=NC/C3/M3/C8** : 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. (Starting with GDAL 1.7.0.)
 - C8 means JPEG2000 compression (one block) and is available for CreateCopy() and/or Create() methods. JPEG2000 compression is only available if the JP2ECW, JP2KAK, JP2OpenJPEG or Jasper driver are available (tried in that order when several ones are available)
 - * *JP2ECW*: The TARGET and PROFILE 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 JP2OpenJPEG-specific creation options can be used. By default BLOCKXSIZE=BLOCKYSIZE=1024 will be used.
 - * Jasper JPEG2000 driver: only in the CreateCopy() case.
- **NUMI=n** : (Starting with GDAL 1.7.0) Number of images. Default = 1. This option is only compatible with IC=NC (uncompressed images).
- **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).

- **FHDR:** File version can be selected though currently the only two variations supported are “NITF02.10” (the default), and “NSIF01.00”.
- **IREP:** Set to “RGB/LUT” to reserve space for a color table for each output band. (Only needed for Create() method, not CreateCopy()).
- **IREPBAND:** (GDAL >= 1.9.0) Comma separated list of band IREPBANDs in band order.
- **ISUBCAT:** (GDAL >= 1.9.0) Comma separated list of band ISUBCATs in band order.
- **LUT_SIZE:** Set to control the size of pseudocolor tables for RGB/LUT bands. A value of 256 assumed if not present. (Only needed for Create() method, not CreateCopy()).
- **BLOCKXSIZE=n:** Set the block width.
- **BLOCKYSIZE=n:** Set the block height.
- **BLOCKA_*=:** 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 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:** (GDAL >= 1.8.0) Similar to above options, except that the TREs are written in the file header, instead of the image header.
- **SDE_TRE=YES/NO:** (GDAL >= 1.8.0) Write GEOLOB and GEOPSB TREs to get more precise georeferencing. This is limited to geographic SRS, and to CreateCopy() for now.
- **RPC00B=YES/NO:** (GDAL >= 2.2.0) 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. Default to YES.
- **RPCTXT=YES/NO:** (GDAL >= 2.2.0) 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. Default to NO.
- **USE_SRC_NITF_METADATA=YES/NO:** (GDAL >= 2.3.0) 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. Default to YES.

4.111.4 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

4.111.5 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.

4.112 NTv1 – NTv1 Datum Grid Shift

Driver short name

NTv1

Driver built-in by default

This driver is built-in by default

NOTE: Implemented as `gdal/frmts/raw/ntv1dataset.cpp`.

4.112.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (vsimem/, etc.)*

4.113 NTv2 – NTv2 Datum Grid Shift

Driver short name

NTv2

Driver built-in by default

This driver is built-in by default

NOTE: Implemented as `gdal/frmts/raw/ntv2dataset.cpp`.

4.113.1 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.*)

4.114 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.

4.114.1 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.)*

4.114.2 Driver capabilities (NWT_GRC)

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

4.114.3 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.

4.114.4 Metadata

GDAL Metadata items are stored in the PAM .aux.xml file

Northwood Grid itself does not natively support arbitrary metadata

4.114.5 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.

4.114.5.1 Creation Options

- **ZMIN=-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=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=50**: Set the brightness level. Only affects opening the file in MapInfo/Vertical Mapper
- **CONTRAST=50**: Set the contrast level. Only affects opening the file in MapInfo/Vertical Mapper
- **TRANSCOLOR=0**: Set a transparent color level. Only affects opening the file in MapInfo/Vertical Mapper
- **TRANSLUCENCY=0**: Set the translucency level. Only affects opening the file in MapInfo/Vertical Mapper

4.115 OZI – OZF2/OZFX3 raster

Driver short name

OZI

Driver built-in by default

This driver is built-in by default

GDAL supports reading OZF2/OZFX3 raster datasets.

Either the image file or the .map file can be passed to GDAL. To retrieve georeferencing, you need to specify the .map file.

4.115.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.115.2 See also

- [Specification of OZF2/OZFX3 format](#)

4.116 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.

4.116.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.116.2 See Also

- [RESTEC Sample Data](#)

4.117 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.

Creation Options:

- **INTERLEAVE=PIXEL/LINE/BAND:** Establish output interleaving, the default is BAND.

NOTE: Implemented as `gdal/frmts/raw/pauxdataset.cpp`.

See Also: [PCI's .aux Format Description](#)

4.117.1 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.)*

4.118 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.

Starting with GDAL 2.0, vector segments are also supported by the driver.

4.118.1 Creation Options

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:** Sets the compression to use. Values other than NONE (the default) 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=n:** When INTERLEAVING is TILED, the tilesize may be selected with this parameter - the default is 127 for 127x127.

4.118.2 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.)*

4.118.3 See Also:

- Implemented as `gdal/frmts/pcidsk/pcidskdataset2.cpp`.
- [PCIDSK SDK](#)

4.119 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_Byte	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 `gdal/frmts/pcraster/pcrasterdataset.cpp`.

See also: [PCRaster website at Utrecht University](#).

4.119.1 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.*)

4.120 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 OGC encoding best practice, or according to the Adobe Supplement to ISO 32000.

Multipage documents are exposed as subdatasets, one subdataset par page of the document.

4.120.1 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.*)

4.120.2 Vector support

See the *PDF vector* documentation page

4.120.3 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.

Starting with GDAL 1.9.0, XMP metadata can be extracted from the file, and will be stored as XML raw content in the xml:XMP metadata domain.

Starting with GDAL 1.10.0, 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.

4.120.4 Configuration options

- *GDAL_PDF_DPI* : To control the dimensions of the raster by specifying the DPI of the rasterization with the Its default value is 150. Starting with GDAL 1.10, 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* : (GDAL >= 1.10.0) 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.
- *GDAL_USER_PWD* : User password for protected PDFs.
- *GDAL_PDF_RENDERING_OPTIONS* : 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 or 4 : whether the PDF should be rendered as a RGB (3) or RGBA (4) image. Defaults to 3.
- *GDAL_PDF_LAYERS* = 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* = 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”.

4.120.4.1 Open Options

Since GDAL 2.0, above configuration options are also available as open options.

- **RENDERING_OPTIONS**=[RASTER,VECTOR,TEXT / RASTER,VECTOR / RASTER,TEXT / RASTER / VECTOR,TEXT / VECTOR / TEXT]: same as GDAL_PDF_RENDERING_OPTIONS configuration option
- **DPI**=value: same as GDAL_PDF_DPI configuration option
- **USER_PWD**=password: same as GDAL_USER_PWD configuration option
- **PDF_LIB**=[POPPLER/PODOFO/PDFIUM]: only available for builds with multiple backends.
- **LAYERS**=string: list of layers (comma separated) to turn ON. Same as GDAL_PDF_LAYERS configuration option
- **GDAL_PDF_LAYERS_OFF**=string: list of layers (comma separated) to turn OFF. Same as GDAL_PDF_LAYERS_OFF configuration option

- **BANDS**=3 or 4. Same as GDAL_PDF_BANDS configuration option
- **NEATLINE**=name of neatline. Same as GDAL_PDF_NEATLINE configuration option

4.120.5 LAYERS Metadata domain

Starting with GDAL >= 1.10.0 and 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"
```

4.120.6 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: starting with GDAL 1.10, some raster-only PDF files (such as some USGS GeoPDF 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).

4.120.7 Creation Issues (GDAL >= 1.10.0)

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. It is also possible to write it according to the OGC Best Practice conventions (but limited to a few datum and projection types).

Note: PDF write support does not require linking to any backend.

4.120.7.1 Creation Options

- **COMPRESS=[NONE/DEFLATE/JPEG/JPEG2000]**: Set the compression to use for raster data. DEFLATE is the default.
- **STREAM_COMPRESS=[NONE/DEFLATE]**: Set the compression to use for stream objects (vector geometries, JavaScript content). DEFLATE is the default.
- **DPI=value**: Set the DPI to use. Default to 72. 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**: (GDAL >= 2.2) Whether the UserUnit setting computed from the DPI ($\text{UserUnit} = \text{DPI} / 72.0$) 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 depend 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]**: Only for DEFLATE compression. Might be set to 2 to use horizontal predictor that can make files smaller (but not always!). 1 is the default.
- **JPEG_QUALITY=[1-100]**: Set the JPEG quality when using JPEG compression. A value of 100 is best quality (least compression), and 1 is worst quality (best compression). The default is 75.
- **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**: By default monoblock files are created. This option can be used to force creation of tiled PDF files.
- **BLOCKXSIZE=n**: Sets tile width, defaults to 256.
- **BLOCKYSIZE=n**: Set tile height, defaults to 256.
- **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=name**: Name for layer where the raster is placed. If specified, the raster will 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 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=name**: Name for layer where the extra content specified with EXTRA_STREAM or EXTRA_IMAGES is placed. If specified, the extra content will be placed into a layer that can be toggled/un-toggled in the “Layer tree” of the PDF reader.
- **MARGIN/LEFT_MARGIN/RIGHT_MARGIN/TOP_MARGIN/BOTTOM_MARGIN=value**: Margin around image in user units.
- **GEO_ENCODING=[NONE/ISO32000/OGC_BP/BOTH]**: Set the Geo encoding method to use. ISO32000 is the default.
- **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]**: 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, CREATOR, CREATION_DATE, KEYWORDS, PRODUCER, SUBJECT, TITLE** : metadata that can be written into the PDF Info block. Note: the format of the value for CREATION_DATE must be D:YYYYMMDDHHmmSSOHH’m’m’ (e.g. D:20121122132447+02’00’ for 22 nov 2012 13:24:47 GMT+02) (see [PDF Reference, version 1.7](#), page 160)
- **OGR_DATASOURCE=name** : Name of the OGR datasource to display on top of the raster layer.
- **OGR_DISPLAY_FIELD=name** : 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=names** : 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** : Whether to write attributes of OGR features. Defaults to YES
- **OGR_LINK_FIELD=name** : 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=names**: 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=names**: Comma separated list of layer names, such that only one of those layers can be visible at a time. This is the behaviour 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=script_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”

4.120.8 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. OGC_BP can be selected by setting the `GDAL_PDF_GEO_ENCODING` configuration option to `OGC_BP`.

Updated elements are written at the end of the file, following the incremental update method described in the PDF specification.

4.120.9 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.

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">
```

(continues on next page)

(continued from previous page)

```

        <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 -->
            <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>
            <IfLayerOn layerId="l2.3">
                <PDF dataset="parcels.pdf">
                    <Blending function="Normal" opacity="0.7"/>
                </PDF>
            </IfLayerOn>
        </IfLayerOn>
    </Content>
</Page>

```

(continues on next page)

(continued from previous 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="11"/>
        <JavascriptAction>app.fs.isFullScreen = true;</JavascriptAction>
      </Actions>
    </OutlineItem>
    <OutlineItem name="Page 1" pageId="page_1">
      <OutlineItem name="Important feature !">
        <Actions>
          <GotoPageAction pageId="page_1" x1="1" y1="2" x2="3" y2="4"/>
        </Actions>
      </OutlineItem>
    </OutlineItem>
    <OutlineItem name="Page 2" pageId="page_2"/>
  </Outline>
</PDFComposition>

```

4.120.10 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, supported since GDAL 2.1.0)

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.

4.120.10.1 Unix build

The relevant configure options are `--with-poppler`, `--with-podofo`, `--with-podofo-lib` and `--with-podofo-extra-lib-for-test`.

Starting with GDAL 2.1.0, `--with-pdfium`, `--with-pdfium-lib`, `--with-pdfium-extra-lib-for-test` and `--enable-pdf-plugin` are also available.

4.120.10.2 Poppler

libpoppler itself must have been configured with `--enable-xpdf-headers` 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.

4.120.10.3 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.

4.120.10.4 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 >= 3.1.0

The scripts in the https://github.com/rouault/pdfium_build_gdal_3_1 repository must be used to build a patched version of PDFium.

With GDAL >= 2.2.0 and < 3.1

A [PDFium forked version for simpler builds](#) is available (for Windows, a dedicated `win_gdal_build` branch is recommended). A [build repository](#) is available with a few scripts that can be used as a template to build PDFium for Linux/MacOSX/Windows. Those forked versions remove the dependency to the V8 JavaScript engine, and have also a few changes to avoid symbol clashes, on Linux, with `libjpeg` and `libopenjpeg`. Building the PDF driver as a GDAL plugin is also a way of avoiding such issues. PDFium build requires a C++11 compatible compiler, as well as for building GDAL itself against PDFium. Successfully tested versions are GCC 4.7.0 (previous versions aren't compatible) and Visual Studio 12 / VS2013.

4.120.11 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/');
```

4.120.12 See also

PDF vector documentation page

Specifications :

- OGC GeoPDF Encoding Best Practice Version 2.2 (08-139r3)
- Adobe Supplement to ISO 32000
- PDF Reference, version 1.7
- Acrobat(R) JavaScript Scripting Reference

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

4.121 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.

4.121.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (vsimem/, etc.)*

4.121.2 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](#). Due to corrections made in [ticket #5941](#) it is recommended to GDAL 1.11.4 or higher.

As a test (for GDAL versions 1.11.4 and higher), 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 in correctly using GDAL (versions 1.11.4 and higher) set 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.

4.121.3 See Also

- Implemented as `gdal/frmts/pds/pdsdataset.cpp`.
- [NASA Planetary Data System](#)
- *ISIS2 – USGS Astrogeology ISIS Cube (Version 2)* driver.
- *ISIS3 – USGS Astrogeology ISIS Cube (Version 3)* driver.

4.122 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 composed 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 of a raw imagery file.

The driver also reads and writes 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.

Starting with GDAL 2.5, the PDS4 driver supports reading and writing ASCII fixed-width, binary fixed-width and delimited(CSV) tables as OGR vector layers.

4.122.1 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.)*

4.122.2 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.

4.122.3 Open options (vector only)

New in version 3.0.

When opening a PDS4 vector dataset, the following open options are available:

- **LAT**=string. Name of a field containing a Latitude value. Defaults to Latitude.
- **LONG**=string. Name of a field containing a Longitude value. Defaults to Longitude.
- **ALT**=string. Name of a field containing a Altitude value. Defaults to Altitude.
- **WKT**=string. Name of a field containing a WKT value.
- **KEEP_GEOM_COLUMNS**=YES/NO. Whether to expose original x/y/geometry columns as regular fields. Defaults to NO.

4.122.4 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.

The following dataset creation options are available:

- Raster only:
 - **IMAGE_FILENAME**=filename. Override default external image filename.
 - **IMAGE_EXTENSION**=ext. 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. Pixel organization in the image file. BSQ is Band SeQuential, BIP is Band Interleaved per Pixel and BIL is Band Interleave Per Line. The default is BSQ. BIL is not valid for `IMAGE_FORMAT=GEOTIFF`
 - **USE_SRC_LABEL**=YES/NO. Whether to use the source label in PDS4 to PDS4 conversions. Defaults to YES.
 - **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. To set the XML element that defines the type of array. Defaults to `Array_3D_Image`. 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 of 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). Consult the [wiki](#) for more information.
 - **LATITUDE_TYPE**=Planetocentric/Planetographic. Value of `latitude_type`. Defaults to Planetocentric.
 - **LONGITUDE_DIRECTION**=Positive East/Positive West. Value of `longitude_direction`. Defaults to Positive East.
 - **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

4.122.5 Layer creation options (vector/table datasets)

(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).
- **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 layer, 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**=string. Name of a field containing a Latitude value. Defaults to Latitude. Only used when the geometry comes from a Point layer with geographic CRS
- **LONG**=string. Name of a field containing a Longitude value. Defaults to Longitude. Only used when the geometry comes from a Point layer with geographic CRS
- **ALT**=string. Name of a field containing a Altitude value. Defaults to Altitude. Only used when the geometry comes from a Point layer with geographic CRS
- **WKT**=string. Name of a field containing a WKT value.
- **SAME_DIRECTORY**=YES/NO. Whether table files should be created in the same directory, or in a subdirectory. Defaults to NO, that is that table files will be created in a subdirectory whose name is the basename of the XML file. For example if creating a “foo.xml” PDS4 dataset, table files will be created in the “foo” subdirectory by default. If this option is set to YES, they will be created in the same directory as “foo.xml”.

4.122.6 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.

4.122.7 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
```

(continues on next page)

(continued from previous page)

```

        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,
    ↪= 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
```

4.122.8 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
```

4.122.9 Limitations

As a new driver and new format, please report any issues to the bug tracker, as explained on the [wiki](#)

4.122.10 See Also:

- Implemented as `gdal/frmts/pds/pds4dataset.cpp`.
- [Official documentation](#)
- [Schemas, including the cartography extension](#)
- *PDS – Planetary Data System v3* driver.
- *ISIS3 – USGS Astrogeology ISIS Cube (Version 3)* driver.

4.123 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.

4.123.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

4.123.2 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.

4.123.3 Open options

The following open options are available : `API_KEY`, `MOSAIC`, `CACHE_PATH`, `TRUST_CACHE` and `USE_TILES`. They have the same semantics as the above describe parameters of same name.

4.123.4 Configuration options

The following configuration options are available :

- **PL_API_KEY**=value: To specify the Planet API key.

4.123.5 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>
```

(continues on next page)

(continued from previous page)

```

    </Scene>
    <Scene>
      <link>https://api.planet.com/data/v1/item-types/PSScene3Band/items/20161010_
↪000309_0e26</link>
    </Scene>
    <Scene>
      <link>https://api.planet.com/data/v1/item-types/PSScene3Band/items/20161119_
↪000452_0e14</link>
    </Scene>
  </Scenes>
</LocationInfo>

```

4.123.5.1 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:

```

(continues on next page)

(continued from previous page)

```

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 = (-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

```

4.123.6 See Also

- [Documentation of Planet Mosaics API](#)
- [API Authentication](#)
- [Vector PLScenes / Planet Scenes API driver](#)

4.124 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.

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).

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 option WORLDFILE is set. All pixel types other than 16bit unsigned will be written as eight bit.

Starting with GDAL 1.9.0, XMP metadata can be extracted from the file, and will be stored as XML raw content in the xml:XMP metadata domain.

4.124.1 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.)*

4.124.2 Color Profile Metadata

Starting with GDAL 1.11, 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,l” format for red primary.)
- SOURCE_PRIMARIES_GREEN (xyY in “x,y,l” format for green primary)
- SOURCE_PRIMARIES_BLUE (xyY in “x,y,l” format for blue primary)
- SOURCE_WHITEPOINT (xyY in “x,y,l” 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:

- **WORLDFILE=YES:** Force the generation of an associated ESRI world file (with the extension .wld). See *World File* section for details.
- **ZLEVEL=n:** Set the amount of time to spend on compression. The default is 6. A value of 1 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 (GDAL >= 2.0)
- **DESCRIPTION=value:** Description, written in a TEXT or iTXt chunk (GDAL >= 2.0)
- **COPYRIGHT=value:** Copyright, written in a TEXT or iTXt chunk (GDAL >= 2.0)
- **COMMENT=value:** Comment, written in a TEXT or iTXt chunk (GDAL >= 2.0)
- **WRITE_METADATA_AS_TEXT=YES/NO:** Whether to write source dataset metadata in TEXT chunks (GDAL >= 2.0)
- **NBITS=1/2/4:** Force number of output bits (GDAL >= 2.1)

NOTE: Implemented as `gdal/frmts/png/pngdataset.cpp`.

PNG support is implemented based on the libpng reference library. More information is available at <http://www.libpng.org/pub/png>.

4.125 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.

Creation Options:

- **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 `gdal/frmts/raw/pnmdataset.cpp`.

4.125.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.126 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.

4.126.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports Georeferencing

This driver supports georeferencing

4.126.2 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** - (GDAL >= 2.3.1) 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 the 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.

4.126.2.1 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.

4.126.2.2 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

4.126.3 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?](#)

4.126.4 Credits

The driver developers

- Jorge Arévalo (jorgearevalo at libregis.org)
- David Zwarg (dzwarg at azavea.com)
- Even Rouault (even.rouault at spatialys.com)

4.126.5 See Also

- [GDAL PostGISRaster driver Wiki](#)
- [PostGIS Raster documentation](#)

4.127 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 exceed 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.prj’/‘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.

4.127.1 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

4.127.2 See Also

- [Racurs company home page](#)
- [PHOTOMOD Lite home page](#)

4.128 Rasdaman GDAL driver

Driver short name

Rasdaman

Build dependencies

raslib

Rasdaman is a raster database middleware offering an SQL-style query language on multi-dimensional arrays of unlimited size, stored in a relational database. See www.rasdaman.org for the open-source code, documentation, etc. Currently rasdaman is under consideration for OSGeo incubation.

In our driver implementation, GDAL connects to rasdaman by defining a query template which is instantiated with the concrete subsetting box upon every access. This allows delivering 2-D cutouts from n-D data sets (such as hyperspectral satellite time series, multi-variable climate simulation data, ocean model data, etc.). In particular, virtual imagery can be offered which is derived on demand from ground truth data. Some more technical details are given below.

The connect string syntax follows the WKT Raster pattern and goes like this:

```

rasdaman: query='select a[$x_lo:$x_hi,$y_lo:$y_hi] from MyImages as a' [tileXSize=1024] [tileYSize=1024]
[host='localhost'] [port=7001] [database='RASBASE'] [user='rasguest'] [password='rasguest']

```

The rasdaman query language (rasql) string in this case only performs subsetting. Upon image access by GDAL, the \$ parameters are substituted by the concrete bounding box computed from the input tile coordinates.

However, the query provided can include any kind of processing, as long as it returns something 2-D. For example, this determines the average of red and near-infrared pixels from the oldest image time series:

```

query='select ( a.red+a.nir ) /2 [$x_lo:$x_hi,$y_lo:$y_hi, 0 ] from SatStack as a'

```

Currently there is no support for reading or writing georeferencing information.

4.128.1 See Also

- [Rasdaman Project](#)

4.129 Rasterlite - Rasters in SQLite DB

Driver short name

Rasterlite

Build dependencies

libsqlite3

The Rasterlite driver allows reading and creating Rasterlite databases.

Those databases can be produced by the utilities of the [rasterlite](#) distribution, such as rasterlite_load, rasterlite_pyramids,

The driver supports reading grayscale, paletted and RGB images stored as GIF, PNG, TIFF or JPEG tiles. The driver also supports reading overviews/pyramids, spatial reference system and spatial extent.

Wavelet compressed tiles are not supported by default by GDAL, unless the *Epsilon - Wavelet compressed images* driver is compiled.

GDAL/OGR must be compiled with OGR SQLite driver support. For read support, linking against spatialite library is not required, but recent enough sqlite3 library is needed to read rasterlite databases. rasterlite library is not required either.

For write support a new table, linking against spatialite library **is** required.

Although the Rasterlite documentation only mentions GIF, PNG, TIFF, JPEG and WAVELET (EPSILON driver) as compression formats for tiles, the driver supports reading and writing internal tiles in any format handled by GDAL. Furthermore, the Rasterlite driver also allow reading and writing as many bands and as many band types as supported by the driver for the internal tiles.

4.129.1 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.*)

4.129.2 Connection string syntax in read mode

Syntax: 'rasterlitedb_name' or 'RASTERLITE:rasterlitedb_name[,table=raster_table_prefix][,minx=minx_val,miny=miny_val,maxx=maxx_val,maxy=maxy_val,level=level_number]

where :

- *rasterlitedb_name* is the filename of the RasterLite DB.
- *raster_table_prefix* is the prefix of the raster table to open. For each raster, there are 2 corresponding SQLite tables, suffixed with *_rasters* and *_metadata*
- *minx_val,miny_val,maxx_val,maxy_val* set a user-defined extent (expressed in coordinate system units) for the raster that can be different from the default extent.
- *level_number* is the level of the pyramid/overview to open, 0 being the base pyramid.

4.129.3 Creation issues

The driver can create a new database if necessary, create a new raster table if necessary and copy a source dataset into the specified raster table.

If data already exists in the raster table, the new data will be added. You can use the `WIPE=YES` creation options to erase existing data.

The driver does not support updating a block in an existing raster table. It can only append new data.

Syntax for the name of the output dataset: 'RASTERLITE:rasterlitedb_name,table=raster_table_prefix' or 'rasterlitedb_name'

It is possible to specify only the DB name as in the later form, but only if the database does not already exist. In that case, the raster table name will be based on the DB name itself.

4.129.3.1 Creation options

- **WIPE** (=NO by default): Set to YES to erase all preexisting data in the specified table
- **TILED** (=YES by default) : Set to NO if the source dataset must be written as a single tile in the raster table
- **BLOCKXSIZE**=n: Sets tile width, defaults to 256.
- **BLOCKYSIZE**=n: Sets tile height, defaults to 256.
- **DRIVER**=[GTiff/GIF/PNG/JPEG/EPSILON/...] : name of the GDAL driver to use for storing tiles. Defaults to GTiff
- **COMPRESS**=[LZW/JPEG/DEFLATE/...] : (GTiff driver) name of the compression method
- **PHOTOMETRIC**=[RGB/YCbCr/...] : (GTiff driver) photometric interpretation
- **QUALITY** : (JPEG-compressed GTiff, JPEG and WEBP drivers) JPEG/WEBP quality 1-100. Defaults to 75
- **TARGET** : (EPSILON driver) target size reduction as a percentage of the original (0-100). Defaults to 96.
- **FILTER** : (EPSILON driver) Filter ID. Defaults to 'daub97lift'.

4.129.4 Overviews

The driver supports building (if the dataset is opened in update mode) and reading internal overviews.

If no internal overview is detected, the driver will try using external overviews (.ovr files).

Starting with GDAL 1.10, options can be used for internal overviews building. They can be specified with the `RASTERLITE_OVR_OPTIONS` configuration option, as a comma separated list of the above creation options. See below examples.

Starting with GDAL 1.10, all resampling methods supported by GDAL overviews are available.

4.129.5 Performance hints

Some of the performance hints of the OGR SQLite driver apply. In particular setting the `OGR_SQLITE_SYNCHRONOUS` configuration option to OFF when creating a dataset or adding overviews might increase performance on some filesystems.

After having added all the raster tables and building all the needed overview levels, it is advised to run :

```
ogrinfo rasterlitedb.sqlite -sql "VACUUM"
```

in order to optimize the database, and increase read performances afterwards. This is particularly true with big rasterlite datasets. Note that the operation might take a long time.

4.129.6 Examples

- Accessing a rasterlite DB with a single raster table :

```
$ gdalinfo rasterlitedb.sqlite -noct
```

Output:

```

Driver: Rasterlite/Rasterlite
Files: rasterlitedb.sqlite
Size is 7200, 7200
Coordinate System is:
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",0.01745329251994328,
        AUTHORITY["EPSG","9122"]],
    AUTHORITY["EPSG","4326"]]
Origin = (-5.000000000000000,55.00000000000000)
Pixel Size = (0.002083333333333,-0.002083333333333)
Metadata:
  TILE_FORMAT=GIF
Image Structure Metadata:
  INTERLEAVE=PIXEL
Corner Coordinates:
Upper Left  ( -5.0000000,  55.0000000) ( 5d 0'0.00"W, 55d 0'0.00"N)
Lower Left  ( -5.0000000,  40.0000000) ( 5d 0'0.00"W, 40d 0'0.00"N)
Upper Right ( 10.0000000,  55.0000000) (10d 0'0.00"E, 55d 0'0.00"N)
Lower Right ( 10.0000000,  40.0000000) (10d 0'0.00"E, 40d 0'0.00"N)
Center      (  2.5000000,  47.5000000) ( 2d30'0.00"E, 47d30'0.00"N)
Band 1 Block=480x480 Type=Byte, ColorInterp=Palette
  Color Table (RGB with 256 entries)

```

- Listing a multi-raster table DB :

```
$ gdalinfo multirasterdb.sqlite
```

Output:

```

Driver: Rasterlite/Rasterlite
Files:
Size is 512, 512
Coordinate System is ``
Subdatasets:
  SUBDATASET_1_NAME=RASTERLITE:multirasterdb.sqlite,table=raster1
  SUBDATASET_1_DESC=RASTERLITE:multirasterdb.sqlite,table=raster1
  SUBDATASET_2_NAME=RASTERLITE:multirasterdb.sqlite,table=raster2
  SUBDATASET_2_DESC=RASTERLITE:multirasterdb.sqlite,table=raster2
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)

```

- Accessing a raster table within a multi-raster table DB:

```
$ gdalinfo RASTERLITE:multirasterdb.sqlite,table=raster1
```

- Creating a new rasterlite DB with data encoded in JPEG tiles :

```
$ gdal_translate -of Rasterlite source.tif RASTERLITE:my_db.sqlite,table=source -
↳co DRIVER=JPEG
```

- Creating internal overviews :

```
$ gdaladdo RASTERLITE:my_db.sqlite,table=source 2 4 8 16
```

- Cleaning internal overviews :

```
$ gdaladdo -clean RASTERLITE:my_db.sqlite,table=source
```

- Creating external overviews in a .ovr file:

```
$ gdaladdo -ro RASTERLITE:my_db.sqlite,table=source 2 4 8 16
```

- Creating internal overviews with options (GDAL 1.10 or later):

```
$ gdaladdo RASTERLITE:my_db.sqlite,table=source 2 4 8 16 --config RASTERLITE_OVR_
↳OPTIONS DRIVER=GTiff,COMPRESS=JPEG,PHOTOMETRIC=YCbCr
```

:

4.129.7 See Also

- [Spatialite and Rasterlite home page](#)
- [Rasterlite manual](#)
- [Rasterlite howto](#)
- [Sample databases](#)
- *OGR SQLite driver*

4.130 RasterLite2 - Rasters in SQLite DB

New in version 2.2.

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.

4.130.1 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.)

4.130.2 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`.

4.130.3 Creation

The driver supports creating new databases from scratch, adding new coverages to an existing database and adding sections to an existing coverage.

4.130.4 Creation options

- **APPEND_SUBDATASET=**YES/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=**string: Coverage name. If not specified, the basename of the output file is used.
- **SECTION=**string: Section name. If not specified, the basename of the output file is used.
- **COMPRESS=**NONE/DEFLATE/LZMA/PNG/CCITTFAX4/JPEG/WEBP/CHARS/JPEG2000: Compression method. Default is NONE. See the [information about supported codecs](#). Note that some codecs may not be available depending on how librasterlite2 has been built.
- **QUALITY=**0 to 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_value. Block width. Defaults to 512.
- **BLOCKYSIZE=**int_value. Block height. Defaults to 512.
- **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. Whether to build automatically build relevant pyramids/overviews. Defaults to NO. Pyramids can be built with the BuildOverviews() / gdaladdo.

4.130.5 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
```

4.130.6 See Also

- [Rasterlite2 home page](#)
- *OGR SQLite driver*

4.131 R – R Object Data Store

Driver short name

R

Driver built-in by default

This driver is built-in by default

The R Object File Format is supported for write access, and limited read access by GDAL. This format is the native format R uses for objects saved with the *save* command and loaded with the *load* command. GDAL supports writing a dataset as an array object in this format, and supports reading files with simple rasters in essentially the same organization. It will not read most R object files.

Currently there is no support for reading or writing georeferencing information.

4.131.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.131.2 Creation Options

- **ASCII=YES/NO**: Produce an ASCII formatted file, instead of binary, if set to YES. Default is NO.
- **COMPRESS=YES/NO**: Produces a compressed file if YES, otherwise an uncompressed file. Default is YES.

See Also:

- [R Project](#)

4.132 RDA (DigitalGlobe Raster Data Access)

Driver short name

RDA

New in version 2.3.

Build dependencies

libcurl

This driver can connect to DigitalGlobe RDA REST API. GDAL/OGR must be built with Curl support in order for the RDA driver to be compiled.

The driver retrieves metadata on graphs and fetches the raster by tiles. Data types byte, uint16, int16, uint32, int32, float32 and float64 are supported.

Any valid graph or template is supported via the DigitalGlobe RDA REST API.

There is no support for overviews.

4.132.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

4.132.2 Dataset name syntax

The minimal syntax to open a datasource is :

```
{ "graphId": "some_value", "nodeId": "another_value" }
```

OR

```
{ "templateId": "some_value", "parameters": { "someparam": "someparamval" } }
```

So a JSon serialized document with 2 attributes graph-id and node-id.

Those values can for example be retrieved from graphs built by [GraphStudio](#).

4.132.3 Connection String options (optional)

- `"options": {"delete-on-close": false}`

can be added to the JSon document to request that cached tiles and metadata are not destroyed at dataset closing. The default, if not specified, is true.

- `"options": {"max-connections": 32}`

can be added to the JSon document to request that cached tiles be fetched using a maximum number of concurrent connections. The default, if not specified, is equal to 8 * number of CPUs.

- `"options": {"advise-read": false}`

can be added to the JSon document to request advise read not be used when reading the dataset. The default, if not specified, is true.

4.132.4 Authentication

Access to the API requires an authentication token. For that, 2 parameters (username, password) must be provided to the driver. They can be retrieved from the below configuration options, or from the `~/.gbdx-config` file.

The access token will be cached in `~/.gdal/rda_cache/authentication.json` and reused from there until its expiration period is reached.

4.132.5 Configuration options

The following configuration options are available :

- **GBDX_AUTH_URL**=value: To specify the OAuth authentication endpoint. Defaults to <https://geobigdata.io/auth/v1/oauth/token/>. If not specified, the `auth_url` parameter from `~/.gbdx-config` will be used if it exists.
- **GBDX_RDA_API_URL**=value: To specify the RDA API endpoint. Defaults to <https://rda.geobigdata.io/v1>. If not specified, the `rda_api_url` parameter from `~/.gbdx-config` will be used if it exists.
- **GBDX_USERNAME**=value: To specify the OAuth user name needed to get to an authentication token. If not specified, the `user_name` parameter from `~/.gbdx-config` must be set.
- **GBDX_PASSWORD**=value: To specify the OAuth user name needed to get to an authentication token. If not specified, the `password` parameter from `~/.gbdx-config` must be set.

4.132.6 ~/.gbdx-config file

This file may be created in the home directory of the user (value of the `$HOME` environment variable on Unix, `$USERPROFILE` on Windows). It can contain values from the above configuration options.

```
[gbdx]
auth_url = https://geobigdata.io/auth/v1/oauth/token/ (optional)
rda_api_url = https://rda.geobigdata.io/v1 (optional)
user_name = value (required)
user_password = value (required)
```

4.132.7 Caching

By default, the authentication token is cached in the `~/.gdal/rda_cache` directory. This directory may be changed with the `RDA_CACHE_DIR` configuration option. By default, dataset metadata and tiles are temporarily cached in `~/.gdal/rda_cache/{graph-id}/{node-id}`, and deleted on dataset closing, unless

```
"options": {"delete-on-close": false}
```

is found in the dataset name.

4.132.8 Open Options

By default, the number of concurrent downloads will be `8*number of CPUs` up to a maximum of 64. The maximum number of concurrent connections can be configured by the `MAXCONNECT` option

4.132.8.1 Examples

- Display metadata, and keep it cached:

```
gdalinfo '{"graphId":
↳ "832050eb7d271d8704c8889369ee0a8a1da82acdee1b20e1700b6d053e94d1fe", "nodeId":
↳ "Orthorectify_hko89y", "options": {"delete-on-close": false}}'
```

```
Driver: RDA/DigitalGlobe Raster Data Access driver
Files: none associated
Size is 9911, 7084
Coordinate System is:
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"]]
Origin = (-84.183163638386631, 33.835018117204456)
Pixel Size = (0.000020885734819, -0.000020885734819)
Metadata:
  ACQUISITION_DATE=2017-04-07T16:25:29.156Z
  CLOUD_COVER=0.0
  GSD=2.325 m
  SAT_AZIMUTH=163.7
  SAT_ELEVATION=58.3
  SENSOR_NAME=8-band (Coastal, Blue, Green, Yellow, Red, Red-edge, NIR1, NIR2)
↳ Multispectral
  SENSOR_PLATFORM_NAME=WV02
  SUN_AZIMUTH=143.5
  SUN_ELEVATION=58.6
Image Structure Metadata:
  INTERLEAVE=PIXEL
Corner Coordinates:
Upper Left  ( -84.1831636,  33.8350181)
Lower Left  ( -84.1831636,  33.6870636)
```

(continues on next page)

(continued from previous page)

```

Upper Right ( -83.9761651, 33.8350181)
Lower Right ( -83.9761651, 33.6870636)
Center      ( -84.0796644, 33.7610408)
Band 1 Block=256x256 Type=UInt16, ColorInterp=Undefined
Band 2 Block=256x256 Type=UInt16, ColorInterp=Blue
Band 3 Block=256x256 Type=UInt16, ColorInterp=Green
Band 4 Block=256x256 Type=UInt16, ColorInterp=Yellow
Band 5 Block=256x256 Type=UInt16, ColorInterp=Red
Band 6 Block=256x256 Type=UInt16, ColorInterp=Undefined
Band 7 Block=256x256 Type=UInt16, ColorInterp=Undefined
Band 8 Block=256x256 Type=UInt16, ColorInterp=Undefined

```

- Extract a subwindow from a dataset:

```

gdal_translate -srcwin 1000 2000 500 500 '{"graphId":
↳ "832050eb7d271d8704c8889369ee0a8alda82acdee1b20e1700b6d053e94dlfe", "nodeId":
↳ "Orthorectify_hko89y"}' out.tif

```

- Materialize a dataset specifying a custom number of concurrent connections:

```

gdal_translate -oo MAXCONNECT=96 '{"graphId":
↳ "832050eb7d271d8704c8889369ee0a8alda82acdee1b20e1700b6d053e94dlfe", "nodeId":
↳ "Orthorectify_hko89y"}' out.tif

```

- Materialize a dataset from a template:

```

gdal_translate '{"templateId": "sample", "parameters": { "imageId": "afa56b05-
↳ 35ad-47d1-bc7f-3e23d220482d"}}' out.tif

```

4.133 RDB - *RIEGL* Database

Driver short name

RDB

New in version 3.1.

Build dependencies

rdblib >= 2.2.0.

GDAL can read *.mpx files in the RDB format, the in-house format used by [RIEGL Laser Measurement Systems GmbH](#) through the RDB library.

The driver relies on the RDB library, which can be downloaded [here](#). The minimum version required of the rdblib is 2.2.0.

4.133.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

4.133.2 Provided Bands

All attributes stored in the RDB, but the coordinates, are provided in bands. Vector attributes are split up into multiple bands. The attributes are currently mapped as follows:

RDB attribute	GDAL Band
riegl.surface_normal[0],	Band 1
riegl.surface_normal[1],	Band 2
riegl.surface_normal[2]	Band 3
riegl.reflectance	Band 4
riegl.amplitude	Band 5
riegl.deviation	Band 6
riegl.point_count	Band 7
riegl.pca_thickness	Band 8
riegl.std_dev	Band 9
riegl.height_center	Band 10
riegl.height_mean	Band 11
riegl.height_min	Band 12
riegl.height_max	Band 13
pixel_linear_sums[0]	Band 14
pixel_linear_sums[1]	Band 15
pixel_linear_sums[2]	Band 16
pixel_square_sums[0]	Band 17
pixel_square_sums[1]	Band 18
pixel_square_sums[2]	Band 19
pixel_square_sums[3]	Band 20
pixel_square_sums[4]	Band 21
pixel_square_sums[5]	Band 22
riegl.voxel_count	Band 23
riegl.id	Band 24
riegl.point_count_grid_cell	Band 25

4.134 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 `gdal/frmts/rik/rikdataset.cpp`.

4.134.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (vsimem/, etc.)*

4.135 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.

4.135.1 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.)*

4.135.2 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.

4.135.3 Open Options

- **RMF_SET_VERTCS**: 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.

4.135.4 Creation Options

- **MTW=ON**: Force the generation of MTW matrix (RSW will be created by default).
- **BLOCKXSIZE=n**: Sets tile width, defaults to 256.
- **BLOCKYSIZE=n**: Set tile height. Tile height defaults to 256.
- **RMFHUGE=NO/YES/IF_SAFER**: Creation of huge RMF file (Supported by GIS Panorama since v11). Defaults to NO.
- **COMPRESS=NONE/LZW/JPEG/RMF_DEM**: (From GDAL 2.4) Compression type. Defaults to NONE. Note: JPEG compression supported only with RGB (3-band) Byte datasets. RMF_DEM compression supported only with Int32 one channel MTW datasets.
- **JPEG_QUALITY**: (From GDAL 2.4) JPEG quality 1-100. Defaults to 75.
- **NUM_THREADS=number_of_threads/ALL_CPUS**: (From GDAL 2.4) Enable multi-threaded compression by specifying the number of worker threads. Default is compression in the main thread.

4.135.5 See Also:

- Implemented as `gdal/frmts/rmf/rmfdataset.cpp`.
- “Panorama” GIS homepage

4.136 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 `gdal/frmts/raw/roipacdataset.cpp`.

4.136.1 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.*)

4.137 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 read-only 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.

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
[ ... ]
  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
```

In some situations, *NITF – National Imagery Transmission Format* 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). In situations where it would not give a good result, you can try to set the `RPFTOC_FORCE_RGBA` environment variable to `TRUE` before opening the subdataset. This will cause the driver to expose the subdataset as a RGBA dataset, instead of a paletted one.

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 `gdal/frmts/nitf/rpftocdataset.cpp`

4.137.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (vsimem/, etc.)*

4.138 RRASTER – R Raster

Driver short name

RRASTER

New in version 2.2.

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. Starting with GDAL 2.3, 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.

Starting with GDAL 2.3, 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.

The following creation options are supported:

- INTERLEAVE=BIP/BIL/BSQ. Respectively band interleaved by pixel, band interleaved by line, band sequential. Default to BIL
- PIXELTYPE=SIGNEDBYTE. To write Byte bands as signed byte instead of unsigned byte.

4.138.1 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.*)

4.138.2 See Also

- Description of the “[rasterfile](#)” format

4.139 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.

4.139.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.139.2 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
- β_0 - open with RADARSAT_2_CALIB:BETA0: prepended to filename
- σ_0 - open with RADARSAT_2_CALIB:SIGMA0: prepended to filename
- γ - 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_Byte/GDT_CInt16, also depending on the type of product selected.

4.139.3 See Also

- RadarSat document RN-RP-51-27.

4.140 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.

4.140.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

4.140.2 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.

4.140.3 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/sla-iw-grd-vh-20150705t064241-20150705t064306-006672-008ea0-002.tiff
      S1A_IW_GRDH_1SDV_20150705T064241_20150705T064306_006672_008EA0_24EE.SAFE/
↪measurement/sla-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

```

```

Driver: SAFE/Sentinel-1 SAR SAFE Product

```

(continues on next page)

(continued from previous page)

```

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
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,
        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

```

- A SLC product with 5 swaths in single pol (the first EW1/HH is selected by default):

```
$ gdalinfo S1A_EW_SLC__1SSH_20150226T010823_20150226T010902_004787_005F2B_E43E.
↳SAFE
```

```
Driver: SAFE/Sentinel-1 SAR SAFE Product
Files: S1A_EW_SLC__1SSH_20150226T010823_20150226T010902_004787_005F2B_E43E.SAFE/
↳manifest.safe
      S1A_EW_SLC__1SSH_20150226T010823_20150226T010902_004787_005F2B_E43E.SAFE/
↳measurement/sla-ew1-slc-hh-20150226t010823-20150226t010859-004787-005f2b-001.
↳tiff
Size is 6871, 14016
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=1, Info=
        (0,0) -> (-26.9158879633399,-76.5938687850829,250.211451298701)
GCP[ 1]: Id=2, Info=

...suppressed output...

GCP[272]: Id=273, Info=
        (6870,14015) -> (-35.4972634588715,-75.5331533717809,0)
Metadata:
  ACQUISITION_START_TIME=2015-02-26T01:08:23.095253
  ACQUISITION_STOP_TIME=2015-02-26T01:09:02.335069
  BEAM_MODE=EW
  BEAM_SWATH=EW1
  FACILITY_IDENTIFIER=ESRIN headquarters
  LINE_SPACING=1.992087e+01
  MISSION_ID=S1A
  MODE=EW
  ORBIT_DIRECTION=ASCENDING
  ORBIT_NUMBER=4787
  PIXEL_SPACING=5.990303e+00
  PRODUCT_TYPE=SLC
  SATELLITE_IDENTIFIER=SENTINEL-1
  SENSOR_IDENTIFIER=SAR
  SWATH=EW1
Subdatasets:
  SUBDATASET_1_NAME=SENTINEL1_DS:S1A_EW_SLC__1SSH_20150226T010823_20150226T010902_
↳004787_005F2B_E43E.SAFE:EW1_HH
  SUBDATASET_1_DESC=Single band with EW1 swath and HH polarization
  SUBDATASET_2_NAME=SENTINEL1_DS:S1A_EW_SLC__1SSH_20150226T010823_20150226T010902_
↳004787_005F2B_E43E.SAFE:EW2_HH
  SUBDATASET_2_DESC=Single band with EW2 swath and HH polarization
  SUBDATASET_3_NAME=SENTINEL1_DS:S1A_EW_SLC__1SSH_20150226T010823_20150226T010902_
↳004787_005F2B_E43E.SAFE:EW3_HH
  SUBDATASET_3_DESC=Single band with EW3 swath and HH polarization
  SUBDATASET_4_NAME=SENTINEL1_DS:S1A_EW_SLC__1SSH_20150226T010823_20150226T010902_
↳004787_005F2B_E43E.SAFE:EW4_HH
```

(continues on next page)

(continued from previous page)

```

SUBDATASET_4_DESC=Single band with EW4 swath and HH polarization
SUBDATASET_5_NAME=SENTINEL1_DS:S1A_EW_SLC__1SSH_20150226T010823_20150226T010902_
→004787_005F2B_E43E.SAFE:EW5_HH
SUBDATASET_5_DESC=Single band with EW5 swath and HH polarization
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0,14016.0)
Upper Right ( 6871.0,    0.0)
Lower Right ( 6871.0,14016.0)
Center      ( 3435.5, 7008.0)
Band 1 Block=7852x1 Type=CInt16, ColorInterp=Undefined
Metadata:
  POLARISATION=HH
  SWATH=EW1

```

4.140.4 Data Calibration

Currently the driver does not apply calibration information.

4.140.5 See Also

- [SAR Formats \(ESA Sentinel Online\)](#)
- [SAFE Specification \(ESA Sentinel Online\)](#)
- *GDAL Sentinel-2 driver*

4.141 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 and ERS data products, including single look complex products; however, it is unlikely to work for non-Radar CEOS products. The simpler *CEOS* driver is often appropriate for these.

This driver will attempt to read 15 lat/long GCPs by sampling the per-scanline CEOS superstructure information. It also captures various pieces of metadata from various header files, including:

```

CEOS_LOGICAL_VOLUME_ID=EERS-1-SAR-MLD
CEOS_PROCESSING_FACILITY=APP
CEOS_PROCESSING_AGENCY=CCRS
CEOS_PROCESSING_COUNTRY=CANADA
CEOS_SOFTWARE_ID=APP 1.62

```

(continues on next page)

(continued from previous page)

```
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) = $HH * \text{conj}(HH)$
- Band 2: Covariance_12 (CFloat32) = $\sqrt{2} * HH * \text{conj}(HV)$
- Band 3: Covariance_13 (CFloat32) = $HH * \text{conj}(VV)$
- Band 4: Covariance_22 (Float32) = $2 * HV * \text{conj}(HV)$
- Band 5: Covariance_23 (CFloat32) = $\sqrt{2} * HV * \text{conj}(VV)$
- Band 6: Covariance_33 (Float32) = $VV * \text{conj}(VV)$

The identities of the bands are also reflected in the metadata.

NOTE: Implemented as `gdal/frmts/ceos2/sar_ceosdataset.cpp`.

4.141.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

4.142 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. Starting with GDAL 2.3, the driver can read compressed .sg-grd-z files that are ZIP archives with .sgrd, .sdatt and .prj files.

The driver supports reading the following SAGA datatypes (in brackets the corresponding GDAL types): BIT (GDT_Byte), BYTE_UNSIGNED (GDT_Byte), BYTE (GDT_Byte), 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_Byte), 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 `gdal/frmts/saga/sagadataset.cpp`.

4.142.1 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.*)

4.143 SDTS – USGS SDTS DEM

Driver short name

SDTS

Driver built-in by default

This driver is built-in by default

GDAL includes support for reading USGS SDTS formatted DEMs. USGS DEMs are always returned with a data type of signed sixteen bit integer, or 32bit float. Projection and georeferencing information is also returned.

SDTS datasets consist of a number of files. Each DEM should have one file with a name like XXXCATD.DDF. This should be selected to open the dataset.

The elevation units of DEMs may be feet or meters. The GetType() method on a band will attempt to return if the units are Feet (“ft”) or Meters (“m”).

NOTE: Implemented as `gdal/frmts/sdts/sdtsdataset.cpp`.

4.143.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (vsimem/, etc.)*

4.144 ESRI ArcSDE Raster

Driver short name

SDE

Build dependencies

ESRI SDE

ESRI ArcSDE provides an abstraction layer over numerous databases that allows the storage of raster data. ArcSDE supports n-band imagery at many bit depths, and the current implementation of the GDAL driver should support as many bands as you can throw at it. ArcSDE supports the storage of LZW, JP2K, and uncompressed data and transparently presents this through its C API SDK.

4.144.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

4.144.2 GDAL ArcSDE Raster driver features

The current driver **supports** the following features:

- **Read** support only.
- GeoTransform information for rasters that have defined it.
- Coordinate reference information.
- Color interpretation (palette for datasets with colormaps, greyscale otherwise).
- Band statistics if ArcSDE has cached them, otherwise GDAL will calculate.
- ArcSDE overview (pyramid) support
- 1 bit, 4 bit, 8 bit, 16 bit, and 32 bit data
- IReadBlock support that maps to ArcSDE's representation of the data in the database.
- ArcSDE 9.1 and 9.2 SDK's. Older versions may also work, but they have not been tested.

The current driver **does not support** the following:

- **Writing** GDAL datasets into the database.
- Large, fast, single-pass reads from the database.
- Reading from "ESRI ArcSDE Raster Catalogs."
- NODATA masks.

4.144.3 Performance considerations

The ArcSDE raster driver currently only supports block read methods. Each call to this method results in a request for a block of raster data for **each** band of data in the raster, and single-pass requests for all of the bands for a block or given area is not currently done. This approach consequently results in extra network overhead. It is hoped that the driver will be improved to support single-pass reads in the near future.

The ArcSDE raster driver should only consume a single ArcSDE connection throughout the lifespan of the dataset. Each connection to the database has an overhead of approximately 2 seconds, with additional overhead that is taken for calculating dataset information. Therefore, usage of the driver in situations where there is a lot of opening and closing of datasets is not expected to be very performant.

Although the ArcSDE C SDK does support threading and locking, the GDAL ArcSDE raster driver does not utilize this capability. Therefore, the ArcSDE raster driver should be considered **not threadsafe**, and sharing datasets between threads will have undefined (and often disastrous) results.

4.144.4 Dataset specification

SDE datasets are specified with the following information:

```
SDE:sdemachine.iastate.edu,5151,database,username,password,fully.specified.tablename,
↳RASTER
```

- **SDE:** – this is the prefix that tips off GDAL to use the SDE driver.
- **sdemachine.iastate.edu** – the DNS name or IP address of the server we are connecting to.
- **5151** – the port number (5151 or port:5151) or service entry (typically *esri_sde*).
- **database** – the database to connect to. This can also be blank and specified as „,

- **username** – username.
- **password** – password.
- **fully.specified.tablename** – It is prudent to use a fully specified tablename wherever possible, although it is not absolutely required.
- **RASTER** – Optional raster column name.

4.145 SENTINEL2 – Sentinel-2 Products

Driver short name

SENTINEL2

Driver built-in by default

This driver is built-in by default

Driver for Sentinel-2 Level-1B, Level-1C and Level-2A products. Starting with GDAL 2.1.3, 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 [Sentinels Scientific Data Hub](#)

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

4.145.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.145.2 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 accros-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.

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.

4.145.3 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.

4.145.4 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, plus the 10m bands, except B8, resampled to 20m, and L2A specific bands (AOT, WVP, SCL, CLD and SNW),
- one for the 3 native 60m bands, plus the 10m&20m bands, except B8, resampled to 60m, and L2A specific bands (AOT, WVP, SCL, CLD and SNW),
- 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.

4.145.5 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.

4.145.6 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 $(\text{subdataset_width} * 2048 * 2) / 10000000$ 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).

4.145.7 Open options

The driver can be passed the following open options:

- **ALPHA=YES/NO**: whether to expose an alpha band. Defaults to NO. 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.

Note: above open options can also be specified as configuration options, by prefixing the open option name with SENTINEL2_ (e.g. SENTINEL2_ALPHA).

4.145.8 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
```

(continues on next page)

(continued from previous page)

```

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_SAFLL1C_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_SAFLL1C_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_SAFLL1C_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_SAFLL1C_PDMC_20150818T101440_R022_
  ↳V20150813T102406_20150813T102406.xml:PREVIEW:EPSG_32632
  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_SAFLL1C_PDMC_20150818T101440_R022_
  ↳V20150813T102406_20150813T102406.xml:10m:EPSG_32632

```

```

Driver: SENTINEL2/Sentinel 2
Files: S2A_OPER_MTD_SAFLL1C_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"]],

```

(continues on next page)

(continued from previous page)

```

    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"]],
    AXIS["Easting",EAST],
    AXIS["Northing",NORTH],
    AUTHORITY["EPSG","32632"]]]
Origin = (699960.0000000000000000,5100060.0000000000000000)
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
[...]
```

- Conversion of a L1C subdataset to tiled GeoTIFF

```

$ gdal_translate SENTINEL2_L1C:S2A_OPER_MTD_SAF_L1C_PDMC_20150818T101440_R022_
→V20150813T102406_20150813T102406.xml:10m:EPSG_32632 \
    10m.tif \
    -co TILED=YES --config GDAL_CACHEMAX 1000 --config GDAL_NUM_
→THREADS 2

```

(continues on next page)

(continued from previous page)

- Generating blank overviews for a L1C subdataset:

```
$ gdaladdo -r NONE SENTINEL2_L1C:S2A_OPER_MTD_SAF_L1C_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):

```
$ python -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_SAF_L1C_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
```

(continues on next page)

(continued from previous page)

```

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
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)

[...]

4.145.9 See Also

- [Sentinels Scientific Data Hub](#)
- [Sentinel 2 User guide](#)
- [Sentinel 2 User Handbook](#)

4.145.10 Credits

This driver has been developed by [Spatialys](#) with funding from [Centre National d’Etudes Spatiales \(CNES\)](#)

4.146 SGI – SGI Image Format

Driver short name

SGI

Driver built-in by default

This driver is built-in by default

The SGI driver currently supports the reading and writing of SGI Image files.

The driver currently supports 1, 2, 3, and 4 band images. The driver currently supports “8 bit per channel value” images. The driver supports both uncompressed and run-length encoded (RLE) images for reading, but created files are always RLE compressed..

The GDAL SGI Driver was based on Paul Bourke’s SGI image read code.

See Also:

- [Paul Bourke’s SGI Image Read Code](#)
- [SGI Image File Format Document](#)

NOTE: Implemented as `gdal/frmts/sgi/sgidataset.cpp`.

4.146.1 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.*)

4.147 SIGDEM – Scaled Integer Gridded DEM

Driver short name

SIGDEM

New in version 2.4.

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 `gdal/frmts/sigdem/sigdemdataset.cpp`.

4.147.1 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.*)

4.148 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 `gdal/frmts/raw/snodasdataset.cpp`.

4.148.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (vsimem/, etc.)*

4.149 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.

Starting with GDAL 1.11, 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 `gdal/frmts/adrg/srpdataset.cpp`.

4.149.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*vsimem*, etc.)

4.150 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 starting with GDAL 2.1.2, `NXXEYYY[.something].hgt`

Starting with GDAL 2.2, 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.

See Also:

- [SRTM documentation](#)
- [SRTM FAQ](#)
- [SRTM data](#)

NOTE: Implemented as `gdal/frmts/srtmhgt/srtmhgtdataset.cpp`.

4.150.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

4.151 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.

4.151.1 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.)*

4.151.2 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.

4.151.3 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`.

4.151.4 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.

4.151.5 See Also

- Implemented as `gdal/frmts/terrigen/terrigendataset.cpp`.
- See [readme.txt](#) for installation and support information.
- [Terragen Terrain File Specification](#).

4.152 TIL – EarthWatch/DigitalGlobe .TIL

Driver short name

TIL

Driver built-in by default

This driver is built-in by default

NOTE: Implemented as `gdal/frmts/raw/tildataset.cpp`.

4.152.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.153 TileDB - TileDB

Driver short name

TileDB

New in version 3.0.

Build dependencies

TileDB

GDAL can read and write TileDB arrays through the TileDB library.

The driver relies on the Open Source TileDB [library](#) (MIT licensed).

4.153.1 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.*)

4.153.2 Creation options

Various creation and open options exists, among them :

- **TILEDDB_CONFIG=config**: A local file with TileDB configuration [options](#)

4.153.3 See Also

- [TileDB home page](#)

4.154 TSX – TerraSAR-X Product

Driver short name

TSX

Driver built-in by default

This driver is built-in by default

NOTE: Implemented as `gdal/frmts/tsx/tsxdataset.cpp`.

4.154.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.155 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 the 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.

4.155.1 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.*)

4.155.2 Creation Issues

GDAL supports export of geographic (and UTM) USGS DEM and CDED data files, including the ability to generate CDED 2.0 50K products to Canadian federal government specifications.

Input data must already be sampled in a geographic or UTM coordinate system. By default the entire area of the input file will be output, but for CDED50K products the output file will be sampled at the production specified resolution and on product tile boundaries.

If the input file has appropriate coordinate system information set, export to specific product formats can take input in different coordinate systems (i.e. from Albers projection to NAD83 geographic for CDED50K production).

Creation Options:

- **PRODUCT=DEFAULT/CDED50K:** When CDED50K is specified, the output file will be forced to adhere to CDED 50K product specifications. The output will always be 1201x1201 and generally a 15 minute by 15 minute tile (though wider in longitude in far north areas).
- **TOPLEFT=long,lat:** For CDED50K products, this is used to specify the top left corner of the tile to be generated. It should be on a 15 minute boundary and can be given in decimal degrees or degrees and minutes (eg. TOPLEFT=117d15w,52d30n).
- **RESAMPLE=Nearest/Bilinear/Cubic/CubicSpline:** Set the resampling kernel used for resampling the data to the target grid. Only has an effect when particular products like CDED50K are being produced. Defaults to Bilinear.
- **DEMLevelCode=integer** DEM Level (1, 2 or 3 if set). Defaults to 1.
- **DataSpecVersion=integer** :Data and Specification version/revision (eg. 1020)
- **PRODUCER=text:** Up to 60 characters to be put into the producer field of the generated file .
- **OriginCode=text:** Up to 4 characters to be put into the origin code field of the generated file (YT for Yukon).
- **ProcessCode=code:** One character to be put into the process code field of the generated file (8=ANUDEM, 9=FME, A=TopoGrid).
- **TEMPLATE=filename:** For any output file, a template file can be specified. A number of fields (including the Data Producer) will be copied from the template file if provided, and are otherwise left blank.
- **ZRESOLUTION=float:** DEM's store elevation information as positive integers, and these integers are scaled using the "z resolution." By default, this resolution is written as 1.0. However, you may specify a different resolution here, if you would like your integers to be scaled into floating point numbers.
- **NTS=name:** NTS Mapsheet name, used to derive TOPLEFT. Only has an effect when particular products like CDED50K are being produced.
- **INTERNALNAME=name:** Dataset name written into file header. Only has an effect when particular products like CDED50K are being produced.

Example: The following would generate a single CDED50K tile, extracting from the larger DEM coverage yk_3arcsec for a tile with the top left corner -117w,60n. The file yk_template.dem is used to set some product fields including the Producer of Data, Process Code and Origin Code fields.

```
gdal_translate -of USGSDEM -co PRODUCT=CDED50K -co TEMPLATE=yk_template.dem \  
-co TOPLEFT=-117w, 60n yk_3arcsec 031a01_e.dem
```

NOTE: Implemented as `gdal/frmts/usgsdem/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.

4.156 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.

4.156.1 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.*)

4.156.2 Metadata

Starting with GDAL 3.1, the VICAR label can be retrieved as JSON-serialized content in the json:VICAR metadata domain.

For example:

```
$ python
from osgeo import gdal
ds = gdal.Open('../autotest/gdrivers/data/test_vicar_truncated.bin')
print(ds.GetMetadata_List('json:VICAR')[0])
{
  "LBLESIZE":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-LINUX",
  "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,
      "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":-1.0000000000000001e+32,
      "DECLINATION":-1.0000000000000001e+32,
      "OFFSET_ANGLE":-1.0000000000000001e+32,
```

(continues on next page)

(continued from previous page)

```

        "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
```

4.156.3 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.b12.16 -al -q

Layer name: binary_prefixes
OGRFeature(binary_prefixes):0
  EphTime (Real) = 127988268.646895
  Exposure (Real) = 40.1072692871094
  COT (Integer) = 28275

```

(continues on next page)

(continued from previous page)

```

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

```

4.156.4 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.

The available creation options are:

- **COORDINATE_SYSTEM_NAME=PLANETOCENTRIC/PLANETOGRAPHIC.** Value of `MAP.COORDINATE_SYSTEM_NAME`. Defaults to `PLANETOCENTRIC`. 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.** Value of `MAP.override`. Defaults to `EAST`. If specified, and `USE_SRC_MAP` is in effect, this will be taken into account to override the source `POSITIVE_LONGITUDE_DIRECTION`.
- **TARGET_NAME=string.** 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.** Whether to use source label in VICAR to VICAR conversions. Defaults to `YES`.
- **LABEL=string.** 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.** Compression method. Default to `NONE`. For maximum interoperability, do not use `BASIC` or `BASIC2` which are not well specified and not always available in VICAR capable applications.

4.156.5 See Also

- Implemented as `gdal/frmts/pds/vicardataset.cpp`.
- [VICAR documentation](#)
- [VICAR file format](#)

4.157 VRT – GDAL Virtual Format

Driver short name

VRT

Driver built-in by default

This driver is built-in by default

4.157.1 Introduction

The VRT driver is a format driver for GDAL 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 VRT format can also describe *Warped VRT* and *Pansharpened VRT*

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.

4.157.2 .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 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 (*Pansharpended VRT*). It may have SRS, GeoTransform, GCPList, Metadata, MaskBand and VRTRasterBand subelements.

```
<VRTDataset rasterXSize="512" rasterYSize="512">
```

4.157.2.1 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 WKT, and valid input to the SetFromUserInput() method (such as well known GEOGCS names, and PROJ.4 format) is also allowed in the SRS element.

```
<SRS dataAxisToSRSAxisMapping="1,2">PROJCS["NAD27 / UTM zone 11N",GEOGCS["NAD27",DATUM["North_American_Datum_1927",SPHEROID["Clarke_1866",6378206.4,294.9786982139006,AUTHORITY["EPSG","7008"]]],AUTHORITY["EPSG","6267"],PRIMEM["Greenwich",0],UNIT["degree",0.0174532925199433],AUTHORITY["EPSG","4267"]],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"]],AUTHORITY["EPSG","26711"]]</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.

```
<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:** (GDAL >= 1.8.0) 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 description 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>
```

- **VRTRasterBand:** This represents one band of a dataset.

4.157.2.2 VRTRasterBand

It will have a dataType attribute with the type of the pixel data associated with this band (use names Byte, UInt16, Int16, UInt32, Int32, Float32, Float64, CInt16, CInt32, CFloat32 or CFloat64) and the band this element represents (1 based). This element may have Metadata, ColorInterp, NoDataValue, HideNoDataValue, ColorTable, GDAL-RasterAttributeTable, Description and MaskBand subelements as well as the various kinds of source elements such as SimpleSource, ComplexSource, etc. 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:** If this element exists a raster band has a nodata value associated with, of the value given as data in the element.

```
<NoDataValue>-100.0</NoDataValue>
```

- **HideNoDataValue:** If this value is 1, the nodata value will not be reported. Essentially, the caller 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 0 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 element 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>
  </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>0.0</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 bands 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: starting with GDAL 2.0, a more general mechanism to specify resampling algorithms can be used. See above paragraph about the ‘resampling’ attribute.
- **ComplexSource:** The *ComplexSource* is derived from the SimpleSource (so it shares the SourceFilename, SourceBand, SrcRect and DestRect 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.
- **KernelFilteredSource:** The *KernelFilteredSource* is a pixel source derived from the Simple Source (so it shares the SourceFilename, SourceBand, SrcRect and DestRect elements, but it also passes the data through a simple filtering kernel specified with the Kernel element.
- **MaskBand:** (GDAL >= 1.8.0) 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.

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.

The shared attribute, added in GDAL 2.0.0, on the SourceFilename indicates whether the dataset should be shared (value is 1) or not (value is 0). The default is 1. If several VRT datasets referring to the same underlying sources are used in a multithreaded context, shared should be set to 0. Alternatively, the VRT_SHARED_SOURCE configuration option can be set to 0 to force non-shared mode.

Some characteristics of the source band can be specified in the optional SourceProperties tag to enable the VRT driver to differ 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.

Starting with GDAL 1.8.0, 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>
```

Starting with GDAL 2.0, 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>
```

Starting with GDAL 2.0, 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>
```

(continues on next page)

(continued from previous page)

```

<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>

```

ComplexSource

Starting with GDAL 1.11, 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.

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:

- Nodata masking
- Color table expansion
- For linear scaling, applying the scale ratio, then scale offset
- For non-linear scaling, apply $(\text{DstMax} - \text{DstMin}) * \text{pow}((\text{SrcValue} - \text{SrcMin}) / (\text{SrcMax} - \text{SrcMin}), \text{Exponent}) + \text{DstMin}$
- 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>
  <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>
  <SrcMin>0</SrcMin>

```

(continues on next page)

(continued from previous page)

```

<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.

```

<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>

```

Starting with GDAL 2.3, 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>

```

4.157.3 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 2.1) 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.

4.157.4 .vrt Descriptions for Raw Files

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.
- The OpenEV tool includes a File menu option to input parameters describing a raw raster file in a GUI and create the corresponding .vrt file.
- 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">
```

(continues on next page)

(continued from previous page)

```

    <ColorInter>Red</ColorInter>
    <SourceFilename relativetoVRT="1">rgb.raw</SourceFilename>
    <ImageOffset>0</ImageOffset>
    <PixelOffset>3</PixelOffset>
    <LineOffset>1200</LineOffset>
  </VRTRasterBand>
  <VRTRasterBand dataType="Byte" band="2" subClass="VRTRawRasterBand">
    <ColorInter>Green</ColorInter>
    <SourceFilename relativetoVRT="1">rgb.raw</SourceFilename>
    <ImageOffset>1</ImageOffset>
    <PixelOffset>3</PixelOffset>
    <LineOffset>1200</LineOffset>
  </VRTRasterBand>
  <VRTRasterBand dataType="Byte" band="3" subClass="VRTRawRasterBand">
    <ColorInter>Blue</ColorInter>
    <SourceFilename relativetoVRT="1">rgb.raw</SourceFilename>
    <ImageOffset>2</ImageOffset>
    <PixelOffset>3</PixelOffset>
    <LineOffset>1200</LineOffset>
  </VRTRasterBand>
</VRTDataset>

```

4.157.5 Creation of VRT Datasets

The VRT driver supports several methods of creating VRT datasets. As of GDAL 1.2.0 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 `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 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 `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" );

```

(continues on next page)

(continued from previous page)

```

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 `SetMetadataItem()` on that band.

```

poVRTDS->GetRasterBand( 1 )->SetMetadataItem( "HideNoDataValue" , "1" );

```

In this example a virtual dataset is created with the `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 `SetMetadata()` 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_Byte, 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 `CreateCopy()` method. The fact that we used `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;

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 );

```

(continues on next page)

(continued from previous page)

```

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 AddBand() method. The options passed to the 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 SetMetadata() / SetMetadataItem() examples.

```

GDALDriver *poDriver = (GDALDriver *) GDALGetDriverByName( "VRT" );
GDALDataset *poVRTDS;

poVRTDS = poDriver->Create( "out.vrt", 512, 512, 0, GDT_Byte, 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_Byte, papszOptions );
CSLDestroy( papszOptions );

delete poVRTDS;

```

4.157.6 Using Derived Bands (with pixel functions in C/C++)

A specialized type of band is a 'derived' band which derives its pixel information from its source bands. With this type of band you must also specify a pixel function, which has the responsibility of generating the output raster. Pixel functions are created by an application and then registered with GDAL using a unique key.

Using derived bands you can create VRT datasets that manipulate bands on the fly without having to create new band files on disk. For example, you might want to generate a band using four source bands from a nine band input dataset (x0, x3, x4, and x8):

```
band_value = sqrt( (x3*x3+x4*x4) / (x0*x8) );
```

You could write the pixel function to compute this value and then register it with GDAL with the name “MyFirstFunction”. Then, the following VRT XML could be used to display this derived band:

```
<VRTDataset rasterXSize="1000" rasterYSize="1000">
  <VRTRasterBand dataType="Float32" band="1" subClass="VRTDerivedRasterBand">
    <Description>Magnitude</Description>
    <PixelFunctionType>MyFirstFunction</PixelFunctionType>
    <SimpleSource>
      <SourceFilename relativeToVRT="1">nine_band.dat</SourceFilename>
      <SourceBand>1</SourceBand>
      <SrcRect xOff="0" yOff="0" xSize="1000" ySize="1000"/>
      <DstRect xOff="0" yOff="0" xSize="1000" ySize="1000"/>
    </SimpleSource>
    <SimpleSource>
      <SourceFilename relativeToVRT="1">nine_band.dat</SourceFilename>
      <SourceBand>4</SourceBand>
      <SrcRect xOff="0" yOff="0" xSize="1000" ySize="1000"/>
      <DstRect xOff="0" yOff="0" xSize="1000" ySize="1000"/>
    </SimpleSource>
    <SimpleSource>
      <SourceFilename relativeToVRT="1">nine_band.dat</SourceFilename>
      <SourceBand>5</SourceBand>
      <SrcRect xOff="0" yOff="0" xSize="1000" ySize="1000"/>
      <DstRect xOff="0" yOff="0" xSize="1000" ySize="1000"/>
    </SimpleSource>
    <SimpleSource>
      <SourceFilename relativeToVRT="1">nine_band.dat</SourceFilename>
      <SourceBand>9</SourceBand>
      <SrcRect xOff="0" yOff="0" xSize="1000" ySize="1000"/>
      <DstRect xOff="0" yOff="0" xSize="1000" ySize="1000"/>
    </SimpleSource>
  </VRTRasterBand>
</VRTDataset>
```

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>
    ...
  </VRTRasterBand>
</VRTDataset>
```

4.157.6.1 Default Pixel Functions

Starting with GDAL 2.2, GDAL provides a set of default pixel functions that can be used without writing new code:

- **real**: extract real part from a single raster band (just a copy if the input is non-complex)
- **imag**: extract imaginary part from a single raster band (0 for non-complex)
- **complex**: make a complex band merging two bands used as real and imag values
- **mod**: extract module from a single raster band (real or complex)
- **phase**: extract phase from a single raster band $[-\pi, \pi]$ (0 or π for non-complex)
- **conj**: computes the complex conjugate of a single raster band (just a copy if the input is non-complex)
- **sum**: sum 2 or more raster bands
- **diff**: computes the difference between 2 raster bands ($b_1 - b_2$)
- **mul**: multiply 2 or more raster bands
- **cmul**: multiply the first band for the complex conjugate of the second
- **inv**: inverse ($1/x$). Note: no check is performed on zero division
- **intensity**: computes the intensity $\text{Re}(x \cdot \text{conj}(x))$ of a single raster band (real or complex)
- **sqr**: perform the square root of a single raster band (real only)
- **log10**: compute the logarithm (base 10) of the abs of a single raster band (real or complex): $\log_{10}(\text{abs}(x))$
- **dB**: perform conversion to dB of the abs of a single raster band (real or complex): $20 \cdot \log_{10}(\text{abs}(x))$
- **dB2amp**: perform scale conversion from logarithmic to linear (amplitude) (i.e. $10^{(x/20)}$) of a single raster band (real only)
- **dB2pow**: perform scale conversion from logarithmic to linear (power) (i.e. $10^{(x/10)}$) of a single raster band (real only)

4.157.6.2 Writing Pixel Functions

To register this function with GDAL (prior to accessing any VRT datasets with derived bands that use this function), an application calls `GDALAddDerivedBandPixelFunc` with a key and a `GDALDerivedPixelFunc`:

```
GDALAddDerivedBandPixelFunc("MyFirstFunction", TestFunction);
```

A good time to do this is at the beginning of an application when the GDAL drivers are registered.

`GDALDerivedPixelFunc` is defined with a signature similar to `IRasterIO`:

@param `papoSources` A pointer to packed rasters; one per source. The datatype of all will be the same, specified in the `eSrcType` parameter.

@param `nSources` The number of source rasters.

@param `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.

@param `nBufXSize` The width of the buffer image into which the desired region is to be read, or from which it is to be written.

@param `nBufYSize` The height of the buffer image into which the desired region is to be read, or from which it is to be written.

@param eSrcType The type of the pixel values in the papoSources raster array.

@param eBufType The type of the pixel values that the pixel function must generate in the pData data buffer.

@param 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.

@param nLineSpace The byte offset from the start of one scanline in pData to the start of the next.

@return CE_Failure on failure, otherwise CE_None.

```
typedef CPLerr
(*GDALDerivedPixelFunc) (void **papoSources, int nSources, void *pData,
                        int nXSize, int nYSize,
                        GDALDataType eSrcType, GDALDataType eBufType,
                        int nPixelSpace, int nLineSpace);
```

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)
{
    int ii, iLine, iCol;
    double pix_val;
    double x0, x3, x4, x8;

    // ---- Init ----
    if (nSources != 4) return CE_Failure;

    // ---- 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);

            pix_val = sqrt((x3*x3+x4*x4)/(x0*x8));

            GDALCopyWords(&pix_val, GDT_Float64, 0,
                        ((GByte *)pData) + nLineSpace * iLine + iCol * nPixelSpace,
                        eBufType, nPixelSpace, 1);
        }
    }

    // ---- Return success ----
    return CE_None;
}
```

4.157.7 Using Derived Bands (with pixel functions in Python)

Starting with GDAL 2.2, in addition to pixel functions written in C/C++ as documented in the [ref gdal_vrttut_derived_c](#) section, it is possible to use pixel functions written in Python. Both [CPython](https://www.python.org/) and [NumPy](http://www.numpy.org/) are requirements at run-time.

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.
- **PixelFunctionArguments** (optional): It is possible to pass arguments to the Python pixel function by defining attributes in the `PixelFunctionArguments` element.
- **PixelFunctionCode** (required if `PixelFunctionType` is of the form “function_name”, ignored otherwise). The in-lined 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)
- **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`

4.157.7.1 Examples

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>
```

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>
  <SimpleSource>
    <SourceFilename relativeToVRT="1">byte2.tif</SourceFilename>
  </SimpleSource>
</VRTRasterBand>
</VRTDataset>
```

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: X/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
    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])-\
```

(continues on next page)

(continued from previous page)

```

        (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)

```

4.157.7.2 Python module path

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.

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 GDAL_VRT_ENABLE_PYTHON configuration option, which can accept 3 values:

- 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 PixelFunctionCode 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_TRUSTED_MODULES configuration option. 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 2 and 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 short-name like “libpython2.7.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/libpython2.7.so”. The same holds on Windows with shortnames like “python27.dll” if accessible through the PATH or full path names like “c:\python27\python27.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. At the time of writing, the order of versions searched is 2.7, 3.5, 3.6, 3.7, 3.8, 3.4, 3.3, 3.2. Enabling debug information (CPL_DEBUG=ON) will show which Python version is used.

4.157.7.3 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 an 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:
    class jit(object):
        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):
    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]):
```

(continues on next page)

(continued from previous page)

```

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)

[illegible]

4.157.8 Warped VRT

A warped VRT is a VRTDataset with subClass="VRTWarpedDataset". It has a GDALWarpOptions element which describe the warping options.

```
<VRTDataset rasterXSize="20" rasterYSize="20" subClass="VRTWarpedDataset">
  <SRS>PROJCS["NAD27 / UTM zone 11N",GEOGCS["NAD27",DATUM["North_American_Datum_1927
  ↳",SPHEROID["Clarke 1866",6378206.4,294.9786982138982,AUTHORITY["EPSG","7008"]],
  ↳AUTHORITY["EPSG","6267"]],PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],UNIT[
  ↳"degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","4267"]],
  ↳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","26711"]]</SRS>
  <GeoTransform> 4.407200000000000e+05, 6.000000000000000e+01, 0.
  ↳0000000000000000e+00, 3.751320000000000e+06, 0.000000000000000e+00, -6.
  ↳0000000000000000e+01</GeoTransform>
  <Metadata>
    <MDI key="AREA_OR_POINT">Area</MDI>
  </Metadata>
  <VRTRasterBand dataType="Byte" band="1" subClass="VRTWarpedRasterBand">
    <ColorInterp>Gray</ColorInterp>
  </VRTRasterBand>
  <BlockXSize>20</BlockXSize>
  <BlockYSize>20</BlockYSize>
  <GDALWarpOptions>
    <WarpMemoryLimit>6.71089e+07</WarpMemoryLimit>
    <ResampleAlg>NearestNeighbour</ResampleAlg>
    <WorkingDataType>Byte</WorkingDataType>
    <Option name="INIT_DEST">0</Option>
    <SourceDataset relativeToVRT="1">byte.vrt</SourceDataset>
    <Transformer>
      <ApproxTransformer>
        <MaxError>0.125</MaxError>
        <BaseTransformer>
          <GenImgProjTransformer>
            <SrcGeoTransform>440720,60,0,3751320,0,-60</SrcGeoTransform>
            <SrcInvGeoTransform>-7345.33333333333303,0.016666666666666664,0,
            ↳62522,0,-0.016666666666666664</SrcInvGeoTransform>
            <DstGeoTransform>440720,60,0,3751320,0,-60</DstGeoTransform>
            <DstInvGeoTransform>-7345.33333333333303,0.016666666666666664,0,
            ↳62522,0,-0.016666666666666664</DstInvGeoTransform>
          </GenImgProjTransformer>
        </BaseTransformer>
      </ApproxTransformer>
    </Transformer>
    <BandList>
      <BandMapping src="1" dst="1" />
    </BandList>
  </GDALWarpOptions>
</VRTDataset>
```

4.157.9 Pansharpened VRT

New in version 2.1.

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.

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.

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>
      <SourceBand>1</SourceBand>
    </PanchroBand>
    <SpectralBand dstBand="1">
      <SourceFilename relativeToVRT="1">multispectral.tif</SourceFilename>
      <SourceBand>1</SourceBand>
    </SpectralBand>
    <SpectralBand dstBand="2">
      <SourceFilename relativeToVRT="1">multispectral.tif</SourceFilename>
      <SourceBand>2</SourceBand>
    </SpectralBand>
    <SpectralBand dstBand="3">
      <SourceFilename relativeToVRT="1">multispectral.tif</SourceFilename>
```

(continues on next page)

(continued from previous page)

```

        <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 behaviour 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 (might be useful if the geotransform are somehow dummy, and the top-left and bottom-right corners of all bands match), 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>

```

(continues on next page)

(continued from previous page)

```

</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">
    <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>

```

4.157.10 Multidimensional VRT

New in version 3.1.

See the dedicated *Multidimensional VRT* page.

4.157.10.1 Multidimensional VRT

New 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="4"/>
    <Dimension name="X" size="3"/>

    <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="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 VRT-Dataset 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>
```

(continues on next page)

(continued from previous page)

```
<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* and *Scale*. 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>
```

(continues on next page)

(continued from previous page)

```
<SourceSlab offset="1,1" count="2,2" step="2,1"/>
<DestSlab offset="2,1"/>
</Source>
```

4.157.11 vrt:// connection string

New 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]
```

For example:

```
vrt://my.tif?bands=3,2,1
```

The only supported option currently is bands. Other may be added in the future.

The effect of this 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`.

4.157.12 Multi-threading issues

Warning: The below section applies to GDAL <= 2.2. Starting with GDAL 2.3, the use of VRT datasets is subject to the standard GDAL dataset multi-threaded rules (that is a VRT dataset handle may only be used by a same thread at a time, but you may open several dataset handles on the same VRT file and use them in different threads)

When using VRT datasets in a multi-threading environment, you should be careful to open the VRT dataset by the thread that will use it afterwards. The reason for that is that the VRT dataset uses `GDALOpenShared` when opening the underlying datasets. So, if you open twice the same VRT dataset by the same thread, both VRT datasets will share the same handles to the underlying datasets.

The shared attribute, added in GDAL 2.0.0, on the `SourceFilename` indicates whether the dataset should be shared (value is 1) or not (value is 0). The default is 1. If several VRT datasets referring to the same underlying sources are used in a multithreaded context, shared should be set to 0. Alternatively, the `VRT_SHARED_SOURCE` configuration option can be set to 0 to force non-shared mode.

4.157.13 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... As of GDAL 2.0, `gdal_translate` and `gdalwarp`, by default, increase the pool size to 450.

4.157.14 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.*)

4.158 WCS – OGC Web Coverage Service

Driver short name

WCS

Build dependencies

libcurl

The optional GDAL WCS driver allows use of a coverage in a WCS server as a raster dataset. GDAL acts as a client to the WCS server.

Accessing a WCS server is accomplished by creating a local service description file that contains one `<WCS_GDAL>` XML element. It is important that there is no spaces or other content outside that element. Starting at version 2.3 the service description file is meant to be managed by the driver in a cache directory. User should control the contents of

the service file using options. The dataset name is `WCS : <URL>`, where the `<URL>` is the URL of the server appended potentially appended with WCS version, coverage, and possibly other parameters. If the WCS version is 2.0.1 further parameters can be given to control how the data model of the coverage is mapped to the GDAL data model.

If the URL does not contain a coverage name, the driver attempts to fetch the capabilities document from the server, parse it, and show the resulting metadata to the user. Coverages are shown as subdatasets. If the URL contains a coverage name as parameter (the key ‘coverage’ can be used irrespective of the WCS version), the driver attempts to fetch the coverage description document from the server, parse it, and create service description file. A small test `GetCoverage` request may be done to obtain details of the served data. If the respective server capabilities file is not cached, it will also be fetched.

With service version 2.0.1 (for which support is available starting at GDAL version 2.3), it may be that the coverage has more than two dimensions. In that case, the driver will append the coverage metadata and show zero bands. At that point, the user must use options to further instruct the driver how to deal with extra dimensions and data fields.

The WCS driver supports WCS versions 1.0.0, 1.1.0, 1.1.1, 1.1.2, and 2.0.1 at basic level (version 0.7 is not supported and support for version 2.0.1 is available starting at GDAL 2.3). Any return format that is a single file, and is in a format supported by GDAL should work. The driver will prefer a format with “tiff” in the name, otherwise it will fallback to the first offered format. However, the user may set the preferred format. Coordinate systems are read from the `DescribeCoverage` result.

4.158.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

4.158.2 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 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. (GDAL 2.3)
- **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. eg. "&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. eg. "&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.

4.158.2.1 Range and dimension subsetting

When WCS version 2.0.1 is used, the range (fields/bands) and the dimension can and/or may need to be subsetting. 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 is wanted in the GDAL dataset.

Range and dimension subsetting must be done via URL parameters since from one coverage it is possible to create more than one different GDAL datasets. In the WCS cache this means that there may be the sets of files related to a 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.

4.158.2.2 Other WCS parameters

The following WCS (version 2.0.1) parameters are recognized besides what has been described above. These all can be set either through options or directly into the URL. The ones in URL take precedence. Note that it is up to the server to support/recognize these.

- MediaType
- UpdateSequence
- GEOTIFF:COMPRESSION
- GEOTIFF:JPEG_QUALITY
- GEOTIFF:PREDICTOR
- GEOTIFF:INTERLEAVE
- GEOTIFF:TILING
- GEOTIFF:TILEWIDTH

4.158.2.3 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.

4.158.2.4 The cache

When the “WCS:<URL>” dataset name syntax is used, the server responses, the service description file, and the metadata files are stored in a cache. Generally, if the needed resource is in the cache, it will be used and no extra calls to the server are done.

The default location of the cache directory is \$HOME/.gdal/wcs_cache

The cache contents can be seen as subdatasets using an empty URL:

```
gdalinfo "WCS:"
```

The cache control options/flags are

- **CACHE=path** Overrides the default cache location.
- **CLEAR_CACHE** The cache is completely initialized and all files are deleted.
- **REFRESH_CACHE** 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.

4.158.2.5 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.

4.158.2.6 Time

Starting with GDAL 1.9.0, 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.

4.158.2.7 Examples

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=1.1.0&
↪coverage=BGS_EMODNET_CentralMed-MC01"
```

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
```

4.158.3 See Also

- OGC WCS Standards

4.159 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) starting with GDAL 1.10 and libwebp 0.1.4.

The WEBP driver can be used as the internal format used by the *Rasterlite - Rasters in SQLite DB* driver.

XMP metadata can be extracted from the file, and will be stored as XML raw content in the xml:XMP metadata domain.

4.159.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports VirtualIO

This driver supports *virtual I/O operations (vsimem/, etc.)*

4.159.2 Creation options

Various creation options exists, among them :

- **QUALITY=n**: 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** (GDAL >= 1.10 and libwebp >= 0.1.4): By default, lossy compression is used. If set to True, lossless compression will be used.

4.159.3 See Also

- [WebP home page](#)

4.160 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.

4.160.1 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.*)

4.160.2 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.

<code><GDAL_WMS></code>	
<code><Service name="WMS"></code>	Define what mini-driver to use, currently sup
<code><Version>1.1.1</Version></code>	WMS version. (optional, defaults to 1.1.1)
<code><ServerUrl>http://onearth.jpl.nasa.gov/wms.cgi?</ServerUrl></code>	WMS server URL. (required)
<code><SRS>EPSG:4326</SRS></code>	Image projection (optional, defaults to EPSG
<code><CRS>CRS:83</CRS></code>	Image projection (optional, defaults to EPSG
<code><ImageFormat>image/jpeg</ImageFormat></code>	Format in which to request data. Paletted for
<code><Transparent>FALSE</Transparent></code>	Set to TRUE to include "transparent=TRUE"
<code><Layers>modis%2Cglobal_mosaic</Layers></code>	A URL encoded, comma separated string of
<code><TiledGroupName>Clementine</TiledGroupName></code>	Comma separated list of layers. (required fo
<code><Styles></Styles></code>	Comma separated list of styles. (optional)
<code><BBoxOrder>xyXY</BBoxOrder></code>	Reorder bbox coordinates arbitrarily. May b
<code></Service></code>	
<code><DataWindow></code>	Define size and extents of the data. (required
<code><UpperLeftX>-180.0</UpperLeftX></code>	X (longitude) coordinate of upper-left corner
<code><UpperLeftY>90.0</UpperLeftY></code>	Y (latitude) coordinate of upper-left corner.
<code><LowerRightX>180.0</LowerRightX></code>	X (longitude) coordinate of lower-right corn
<code><LowerRightY>-90.0</LowerRightY></code>	Y (latitude) coordinate of lower-right corner
<code><SizeX>2666666</SizeX></code>	Image size in pixels.
<code><SizeY>1333333</SizeY></code>	Image size in pixels.
<code><TileX>0</TileX></code>	Added to tile X value at highest resolution. (
<code><TileY>0</TileY></code>	Added to tile Y value at highest resolution. (
<code><TileLevel>0</TileLevel></code>	Tile level at highest resolution. (tiled image
<code><TileCountX>0</TileCountX></code>	Can be used to define image size, SizeX = T
<code><TileCountY>0</TileCountY></code>	Can be used to define image size, SizeY = T
<code><YOrigin>top</YOrigin></code>	Can be used to define the position of the Y o

</DataWindow>	
<Projection>EPSG:4326</Projection>	Image projection (optional, defaults to value)
<IdentificationTolerance>2</IdentificationTolerance>	Identification tolerance (optional, defaults to value)
<BandsCount>3</BandsCount>	Number of bands/channels, 1 for grayscale color
<DataType>Byte</DataType>	Band data type, among Byte, Int16, UInt16, Int32, UInt32, Float32, Float64
<DataValues NoData="0 0 0" min="1 1 1" max="255 255 255" />	Define NoData and/or minimum and/or maximum values
<BlockSizeX>1024</BlockSizeX>	Block size in pixels. (optional, defaults to 1024)
<BlockSizeY>1024</BlockSizeY>	Block size in pixels. (optional, defaults to 1024)
<OverviewCount>10</OverviewCount>	Count of reduced resolution layers each having half the resolution
<Cache>	Enable local disk cache. Allows for offline operation
<Path>./gdalwmscache</Path>	Location where to store cache files. It is safe to use relative paths
<Depth>2</Depth>	Number of directory layers. 2 will result in files like: cache/0/1/2/3/4/5/6/7/8/9/
<Extension>.jpg</Extension>	Append to cache files. (optional, defaults to .tif)
<Type>file</Type>	Cache type. Now supported only 'file' type.
<Expires>604800</Expires>	Time in seconds cached files will stay valid.
<MaxSize>67108864</MaxSize>	The cache maximum size in bytes. If cache is full, oldest files are removed
<Unique>True</Unique>	If set to true the path will be appended with md5 hash of the request
</Cache>	
<MaxConnections>2</MaxConnections>	Maximum number of simultaneous connections
<Timeout>300</Timeout>	Connection timeout in seconds. (optional, defaults to 300)
<OfflineMode>true</OfflineMode>	Do not download any new images, use only the cache
<AdviseRead>true</AdviseRead>	Enable AdviseRead API call - download image data in chunks
<VerifyAdviseRead>true</VerifyAdviseRead>	Open each downloaded image and do some verification
<ClampRequests>false</ClampRequests>	Should requests, that otherwise would be partially outside the image, be clamped
<UserAgent>GDAL WMS driver (http://www.gdal.org/frmt_wms.html)</UserAgent>	HTTP User-agent string. Some servers might require it
<UserPwd>user:password</UserPwd>	User and Password for HTTP authentication
<UnsafeSSL>true</UnsafeSSL>	Skip SSL certificate verification. May be needed for some servers
<Referer>http://example.foo/</Referer>	HTTP Referer string. Some servers might require it
<ZeroBlockHttpCodes>204,404</ZeroBlockHttpCodes>	Comma separated list of HTTP response codes that should be treated as zero blocks
<ZeroBlockOnServerException>true</ZeroBlockOnServerException>	Whether to treat a Service Exception returned by the server as a zero block
</GDAL_WMS>	

4.160.3 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.

4.160.3.1 WMS

Communications with an OGC WMS server. Has support for both tiled and untiled requests.

Starting with GDAL >= 1.10, 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&
```

(continues on next page)

(continued from previous page)

```

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>
            </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>

```

4.160.3.2 TileService

Service to support talking to a WorldWind [TileService](#). Access is always tile based.

4.160.3.3 WorldWind

Access to web-based WorldWind tile services. Access is always tile based.

4.160.3.4 TMS (GDAL 1.7.0 and later)

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.

4.160.3.5 OnEarth Tiled WMS (GDAL 1.9.0 and later)

The OnEarth Tiled WMS minidriver supports the Tiled WMS specification implemented for the JPL OnEarth driver per the specification at <http://onearth.jpl.nasa.gov/tiled.html>.

A typical OnEarth Tiled WMS configuration file might look like:

```
<GDAL_WMS>
  <Service name="TiledWMS">
    <ServerUrl>http://onmoon.jpl.nasa.gov/wms.cgi?</ServerUrl>
    <TiledGroupName>Clementine</TiledGroupName>
  </Service>
</GDAL_WMS>
```

Most of the other information is automatically fetched from the remote server using the GetTileService method at open time.

4.160.3.6 VirtualEarth (GDAL 1.9.0 and later)

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 = 19
- OverviewCount = 18
- SRS = EPSG:900913
- BlockSizeX = 256
- BlockSizeY = 256

4.160.3.7 ArcGIS REST API (GDAL 2.0 and later)

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>http://
↪ sampleserver1.arcgisonline.com/ArcGIS/rest/services/Specialty/ESRI_StateCityHighway_
↪ USA/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
```

4.160.3.8 Internet Imaging Protocol (IIP) (GDAL 2.1 and later)

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”

4.160.4 Examples

- [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.

- [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.

```
gdal_translate -of PNG -outsize 500 250 metacarta_wmsc.xml metacarta_wmsc.png

.. only:: html

|example output 1|
```

- [tileservice_bmng.xml](#) - TileService, Blue Marble NG (January)

```
gdal_translate -of JPEG -outsize 500 250 tileservice_bmng.xml tileservice_bmng.
↪ jpg

.. only:: html

|example output 2|
```

- [tileservice_nysdop2004.xml](#) - TileService, NYSDOP 2004

```
gdal_translate -of JPEG -projwin -73.687030 41.262680 -73.686359 41.262345 -
↪ outsize 500 250 tileservice_nysdop2004.xml tileservice_nysdop2004.jpg

.. only:: html

|example output 3|
```

- [OpenStreetMap TMS Service Example](#): 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](#), accessing the default MetaCarta TMS layer.

```
gdal_translate -of PNG -outsize 512 256 frmt_wms_metacarta_tms.xml
metacarta.png
```
- [BlueMarble Amazon S3 Example](#) accessed with the TMS minidriver.
- [Google Maps](#) accessed with the TMS minidriver.
- [ArcGIS MapServer Tiles](#) accessed with the TMS minidriver.
- [OnEarth Tiled WMS Clementine](#), [daily](#), and [srtm](#) examples.
- [VirtualEarth Aerial Layer](#) accessed with the VirtualEarth minidriver.
- [ArcGIS online sample server layer](#) accessed with the ArcGIS Server REST API minidriver.
- [IIP online sample server layer](#) accessed with the IIP minidriver.

4.160.5 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>http://onearth.jpl.nasa.
↪gov/wms.cgi?</ServerUrl><TiledGroupName>Global SRTM Elevation</TiledGroupName></
↪Service></GDAL_WMS>"
```

- (GDAL >= 1.9.0) 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.

- (GDAL >= 1.9.0) 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"
```

- (GDAL >= 1.9.0) the base URL of a Tiled WMS service, prefixed with *WMS:* and with request=GetTileService as GET argument:

```
gdalinfo "WMS:http://onearth.jpl.nasa.gov/wms.cgi?request=GetTileService"
```

A list of subdatasets will be returned, resulting from the parsing of the GetTileService request on that server.

- (GDAL >= 2.0.0) 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"
```

- (GDAL >= 2.1.0) the URL of a IIP image:

```
gdalinfo "IIP:http://merovingio.c2rmf.cnrs.fr/fcgi-bin/iipsrv.fcgi?FIF=globe.  
↪256x256.tif"
```

4.160.6 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.

4.160.7 See Also

- [OGC WMS Standards](#)
- [WMS Tiling Client Recommendation \(WMS-C\)](#)
- [WorldWind TileService](#)
- [TMS Specification](#)
- [OnEarth Tiled WMS specification](#)
- [ArcGIS Server REST API](#)

4.161 WMTS – OGC Web Map Tile Service

Driver short name

WMTS

New in version 2.1.

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.

4.161.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.161.2 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=level]]`

```
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 URL (required), LAYER, TILEMATRIXSET, TILEMATRIX, ZOOM_LEVEL, STYLE, EXTENDBEYONDDATELINE.

```
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.

4.161.3 Local service description XML file

It is important that there be no spaces or other content before the `<GDAL_WMTS>` element.

<code><GDAL_WMTS></code>	
<code><GetCapabilitiesUrl>http://foo/WM TSCapabili- ties.xml</GetCapabiliti esUrl></code>	URL (or filename for local files) to GetCapabilities re- sponse document (required). For a KVP only server, will be like http://end_point?SERVICE=WMTS& REQUEST=GetCapabilities .
<code><Layer>layer_id</Layer></code>	Layer identifier (optional, but may be needed to disam- biguate between several layers)
<code><Style>style_id</Style></code>	Style identifier. Must be one of the listed styles for the layer. (optional, but may be needed to disambiguate be- tween several styles)
<code><TileMatrixSet>tile_matrix_set_id </TileMatrixSet></code>	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)
<code><TileMatrix>tile_matrix_id</TileM atrix></code>	Tile Matrix identifier. Must be one of the listed tile ma- trix of the select tile matrix set for the layer. (optional, GDAL >= 2.2. Exclusive with ZoomLevel. If not spec- ified the last tile matrix, ie the one with the best resolu- tion, is selected)
<code><ZoomLevel>int_value</ZoomLevel></code>	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)
<code><Format>image/png</Format></code>	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)
<code><InfoFormat>application/xml</Info Format></code>	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 Info- Format)
<code><DataWindow></code>	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)
<code><UpperLeftX>-180.0</UpperLeftX></code>	X (longitude/easting) coordinate of upper-left corner, in the SRS of the tile matrix set. (required if DataWindow is present)
<code><UpperLeftY>90.0</UpperLeftY></code>	Y (latitude/northing) coordinate of upper-left corner, in the SRS of the tile matrix set. (required if DataWindow is present)
<code><LowerRightX>180.0</LowerRightX></code>	X (longitude/easting) coordinate of lower-right corner, in the SRS of the tile matrix set. (required if DataWin- dow is present)
<code><LowerRightY>-90.0</LowerRightY></code>	Y (latitude/northing) coordinate of lower-right corner, in the SRS of the tile matrix set. (required if DataWin- dow is present)
<code></DataWindow></code>	

Continued on next page

Table 7 – continued from previous page

<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)
<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): <ul style="list-style-type: none"> • 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 <ul style="list-style-type: none"> – 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, absent by default, but enabled in autogenerated XML)
<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)
<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://www.gdal.org/frmt_wms.htm)</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://www.gdal.org/frmt_wms.htm)”).
<UserPwd>user:password</UserPwd>	User and Password for HTTP authentication (optional).

Continued on next page

Table 7 – continued from previous page

<code><UnsafeSSL>true</UnsafeSSL></code>	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).
<code><Referer>http://example.foo/</Ref erer></code>	HTTP Referer string. Some servers might require it (optional).
<code><ZeroBlockHttpCodes>204,404</Zero BlockHttpCodes></code>	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)
<code><ZeroBlockOnServerException>true< /ZeroBlockOnServerException></code>	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)
<code></GDAL_WMTS></code>	

4.161.4 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
```

4.161.5 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>
    <LowerRightY>6123507.385072636</LowerRightY>
  </DataWindow>
  <BandsCount>4</BandsCount>
  <Cache />
  <UnsafeSSL>true</UnsafeSSL>
  <ZeroBlockHttpCodes>404</ZeroBlockHttpCodes>
```

(continues on next page)

(continued from previous page)

```
<ZeroBlockOnServerException>true</ZeroBlockOnServerException>
</GDAL_WMTS>
```

The generated file will come with default values that you may need to edit.

4.161.6 See Also

- [OGC WMTS Standard](#)
- [WMS – Web Map Services](#) driver page.

4.162 XPM – X11 Pixmap

Driver short name

XPM

Driver built-in by default

This driver is built-in by default

GDAL includes support for reading and writing XPM (X11 Pixmap Format) image files. These are colormapped one band images primarily used for simple graphics purposes in X11 applications. It has been incorporated in GDAL primarily to ease translation of GDAL images into a form usable with the GTK toolkit.

The XPM support does not support georeferencing (not available from XPM files) nor does it support XPM files with more than one character per pixel. New XPM files must be colormapped or greyscale, and colortables will be reduced to about 70 colors automatically.

NOTE: Implemented as `gdal/frmts/xpm/xpmdataset.cpp`.

4.162.1 Driver capabilities

Supports CreateCopy()

This driver supports the `GDALDriver::CreateCopy()` operation

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

4.163 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.

The spacing between each cell must be constant and no missing value is 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. 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 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.

4.163.1 Creation options

- **COLUMN_SEPARATOR**=a_value : where a_value is a string used to separate the values of the X,Y and Z columns. Defaults to one space character
- **ADD_HEADER_LINE**=YES/NO : whether an header line must be written (content is X <col_sep> Y <col_sep> Z) . Defaults to NO
- **SIGNIFICANT_DIGITS**=a_value : where a_value specifies the number of significant digits to output (%g format; is defaults with 18)
- **DECIMAL_PRECISION**=a_value : where a_value specifies the number of decimal places to output when writing floating-point numbers (%f format; alternative to SIGNIFICANT_DIGITS).

4.163.2 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.*)

4.163.3 See also

- Documentation of *gdal_grid*

4.164 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 `gdal/frmts/zmap/zmapdataset.cpp`.

4.164.1 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.*)

VECTOR DRIVERS

Short name	Long name	Creation	Geo-referencing	Build requirements
<i>AERONAVFAA</i>	Aeronav FAA	No	Yes	Built-in by default
<i>AmigoCloud</i>	AmigoCloud	Yes	Yes	libcurl
<i>AO</i>	ESRI ArcObjects	No	No	ESRI ArcObjects
<i>ARCGEN</i>	Arc/Info Generate	No	Yes	Built-in by default
<i>AVCBIN</i>	Arc/Info Binary Coverage	No	Yes	Built-in by default
<i>AVCE00</i>	Arc/Info E00 (ASCII) Coverage	No	Yes	Built-in by default
<i>BNA</i>	Atlas BNA	Yes	Yes	Built-in by default
<i>CAD</i>	AutoCAD DWG	No	Yes	(internal libopencad provided)
<i>CARTO</i>	Carto	Yes	Yes	libcurl
<i>Cloudant</i>	Cloudant	Yes	Yes	libcurl
<i>CouchDB</i>	CouchDB/GeoCouch	Yes	Yes	libcurl
<i>CSV</i>	Comma Separated Value (.csv)	Yes	Yes	Built-in by default
<i>CSW</i>	OGC CSW (Catalog Service for the Web)	No	Yes	libcurl
<i>DB2</i>	DB2 Spatial	Yes	Yes	ODBC library
<i>DGN</i>	Microstation DGN	Yes	Yes	Built-in by default
<i>DGNv8</i>	Microstation DGN v8	Yes	Yes	Open Design Alliance Teigha library
<i>DODS</i>	DODS/OPeNDAP	No	Yes	libdap
<i>DWG</i>	AutoCAD DWG	No	No	Open Design Alliance Teigha library
<i>DXF</i>	AutoCAD DXF	Yes	No	Built-in by default
<i>EDIGEO</i>	EDIGEO	No	Yes	Built-in by default
<i>EEDA</i>	Google Earth Engine Data API	No	Yes	libcurl
<i>Elasticsearch</i>	Elasticsearch: Geographically Encoded Objects for Elasticsearch	Yes	Yes	libcurl
<i>ESRIJSON</i>	ESRIJSON / FeatureService driver	No	Yes	Built-in by default
<i>FileGDB</i>	ESRI File Geodatabase (FileGDB)	Yes	Yes	FileGDB API library
<i>FlatGeobuf</i>	FlatGeobuf	Yes	Yes	Built-in by default
<i>FME</i>	FMEObjects Gateway	No	No	FME

Continued on next page

Table 1 – continued from previous page

Short name	Long name	Creation	Geo-referencing	Build requirements
<i>Geoconcept</i>	GeoConcept text export	Yes	Yes	Built-in by default
<i>GeoJSON</i>	GeoJSON	Yes	Yes	Built-in by default
<i>GeoJSONSeq</i>	GeoJSONSeq: sequence of GeoJSON features	Yes	Yes	Built-in by default
<i>Geomedia</i>	Geomedia MDB database	No	Yes	ODBC library
<i>GeoRSS</i>	GeoRSS : Geographically Encoded Objects for RSS feeds	Yes	Yes	(read support needs libexpat)
<i>GML</i>	Geography Markup Language	Yes	Yes	(read support needs Xerces or libexpat)
<i>GMLAS</i>	Geography Markup Language (GML) driven by application schemas	No	Yes	Xerces
<i>GMT</i>	GMT ASCII Vectors (.gmt)	Yes	Yes	Built-in by default
<i>GPKG</i>	GeoPackage vector	Yes	Yes	libsqlite3
<i>GPSBabel</i>	GPSBabel	Yes	Yes	(read support needs GPX driver and libexpat)
<i>GPX</i>	GPS Exchange Format	Yes	Yes	(read support needs libexpat)
<i>GRASS</i>	GRASS Vector Format	No	Yes	libgrass
<i>GTM</i>	GPS TrackMaker	Yes	Yes	Built-in by default
<i>HTF</i>	Hydrographic Transfer Format	No	Yes	Built-in by default
<i>IDB</i>	IDB	Yes	Yes	Informix DataBlade
<i>IDRISI</i>	Idrisi Vector (.VCT)	No	Yes	Built-in by default
<i>INTERLIS 1</i>	“INTERLIS 1” and “INTERLIS 2” drivers	Yes	Yes	Xerces
<i>INTERLIS 2</i>	“INTERLIS 1” and “INTERLIS 2” drivers	Yes	Yes	Xerces
<i>INGRES</i>	INGRES	Yes	No	INGRESS
<i>JML</i>	JML: OpenJUMP JML format	Yes	Yes	(read support needs libexpat)
<i>KML</i>	Keyhole Markup Language	Yes	Yes	(read support needs libexpat)
<i>LIBKML</i>	LIBKML Driver (.kml .kmz)	Yes	Yes	libkml
<i>MapML</i>	MapML	Yes	Yes	Built-in by default
<i>MDB</i>	Access MDB databases	No	No	JDK/JRE and Jackcess
<i>Memory</i>	Memory	Yes	Yes	Built-in by default
<i>MITAB</i>	MapInfo TAB and MIF/MID	Yes	Yes	Built-in by default
<i>MongoDB</i>	MongoDB	Yes	Yes	Mongo C++ client legacy library
<i>MongoDBv3</i>	MongoDBv3	Yes	Yes	Mongo CXX >= 3.4.0 client library
<i>MSSQLSpatial</i>	Microsoft SQL Server Spatial Database	Yes	Yes	ODBC library
<i>MVT</i>	MVT: Mapbox Vector Tiles	Yes	Yes	(requires SQLite and GEOS for write support)
<i>MySQL</i>	MySQL	Yes	Yes	MySQL library
<i>NAS</i>	ALKIS	No	Yes	Xerces
<i>netCDF</i>	Vector	Yes	Yes	libnetcdf
<i>NGW</i>	NextGIS Web	No	Yes	libcurl
<i>UK .NTF</i>	UK .NTF	No	Yes	Built-in by default
<i>OAPIF</i>	OGC API - Features	No	Yes	libcurl
<i>OCI</i>	Oracle Spatial	Yes	Yes	OCI library

Continued on next page

Table 1 – continued from previous page

Short name	Long name	Creation	Geo-referencing	Build requirements
<i>ODBC</i>	ODBC RDBMS	No	Yes	ODBC library
<i>ODS</i>	Open Document Spreadsheet	Yes	No	libexpat
<i>OGDI</i>	OGDI Vectors	No	Yes	OGDI library
<i>OpenAir</i>	OpenAir Special Use Airspace Format	No	Yes	Built-in by default
<i>OpenFileGDB</i>	ESRI File Geodatabase (OpenFileGDB)	No	Yes	Built-in by default
<i>OSM</i>	OpenStreetMap XML and PBF	No	Yes	libsqlite3 (and libexpat for OSM XML)
<i>PDF</i>	Geospatial PDF	Yes	Yes	none for write support, Poppler/PoDoFo/PDFium for read support
<i>PDS</i>	Planetary Data Systems TABLE	No	No	Built-in by default
<i>PostgreSQL</i>	PostgreSQL / PostGIS	Yes	Yes	PostgreSQL client library (libpq)
<i>PGDump</i>	PostgreSQL SQL Dump	Yes	Yes	Built-in by default
<i>PGeo</i>	ESRI Personal GeoDatabase	No	Yes	ODBC library
<i>PLScenes</i>	PLScenes (Planet Labs Scenes/Catalog API)	No	No	libcurl
<i>S57</i>	IHO S-57 (ENC)	No	Yes	Built-in by default
<i>SDE</i>	ESRI ArcSDE	No	No	ESRI SDE
<i>SDTS</i>	SDTS	No	Yes	Built-in by default
<i>SEGUKOOA</i>	SEG-P1 / UKOOA P1/90	No	Yes	Built-in by default
<i>SEGY</i>	SEG-Y / SEG-Y	No	Yes	Built-in by default
<i>Selafin</i>	Selafin files	Yes	Yes	Built-in by default
<i>ESRI Shapefile</i>	ESRI Shapefile / DBF	Yes	Yes	Built-in by default
<i>SOSI</i>	Norwegian SOSI Standard	No	No	FYBA library
<i>SQLite</i>	SQLite / Spatialite RDBMS	Yes	Yes	libsqlite3 or libspatialite
<i>SUA</i>	Tim Newport-Peace's Special Use Airspace Format	No	Yes	Built-in by default
<i>SVG</i>	Scalable Vector Graphics	No	Yes	libexpat
<i>SXF</i>	SXF	No	Yes	Built-in by default
<i>TIGER</i>	U.S. Census TIGER/Line	No	Yes	Built-in by default
<i>TopoJSON</i>	TopoJSON driver	No	Yes	Built-in by default
<i>VDV</i>	VDV-451/VDV-452/INTREST Data Format	Yes	Yes	Built-in by default
<i>VFK</i>	Czech Cadastral Exchange Data Format	No	Yes	libsqlite3
<i>VRT</i>	Virtual Format	Yes	Yes	Built-in by default
<i>Walk</i>	Walk Spatial Data	No	Yes	ODBC library
<i>WAsP</i>	WAsP .map format	Yes	Yes	Built-in by default
<i>WFS</i>	OGC WFS service	No	Yes	libcurl
<i>XLS</i>	MS Excel format	No	No	libfreexl
<i>XLSX</i>	MS Office Open XML spreadsheet	Yes	No	libexpat
<i>XPlane</i>	X-Plane/Flightgear aeronautical data	No	Yes	Built-in by default

5.1 Aeronav FAA

Driver short name

AERONAVFAA

Driver built-in by default

This driver is built-in by default

This driver reads text files describing aeronav information - obstacles, navaids and routes - as provided by the FAA.

5.1.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (vsimem/, etc.)*

5.1.2 See Also

- Digital Obstacle File
- NAVAID Digital Data File
- Digital Aeronautical Chart Supplement

5.2 AmigoCloud

New in version 2.1.0.

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.

5.2.1 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.)*

5.2.2 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
```

5.2.3 Configuration options

The following configuration options are available :

- **AMIGOCLOUD_API_URL**: defaults to <https://www.amigocloud.com/api/v1>. Can be used to point to another server.
- **AMIGOCLOUD_API_KEY**: see following paragraph.

5.2.4 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.

5.2.5 Geometry

The OGR driver will report as many geometry fields as available in the layer, following RFC 41.

5.2.6 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()`.

5.2.7 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 :

- `OGRFeature::CreateFeature()` \iff INSERT operation
- `OGRFeature::SetFeature()` \iff UPDATE operation
- `OGRFeature::DeleteFeature()` \iff DELETE operation
- `OGRDataSource::CreateLayer()` \iff CREATE TABLE operation
- `OGRDataSource::DeleteLayer()` \iff 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.

5.2.8 Layer creation options

The following layer creation options are available:

- **OVERWRITE=YES/NO**: Whether to overwrite an existing table with the layer name to be created. Defaults to NO.
- **GEOMETRY_NULLABLE=YES/NO**: Whether the values of the geometry column can be NULL. Defaults to YES.

5.2.9 Examples

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"
```

Show list of datasets.

```
$ ogrinfo -ro "AmigoCloud:1234 datasets"
List of available datasets for project id: 1234
| id          | name
|-----|-----
| 5551        | points
| 5552        | lines
```

Accessing data from a list of datasets:

```
ogrinfo -ro "AmigoCloud:1234 datasets=1234,1235"
```

Creating and populating a table from a shapefile:

```
ogr2ogr -f AmigoCloud "AmigoCloud:1234" myshapefile.shp
```

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
```

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
```

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
```

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
```

5.2.10 See Also

- [AmigoCloud API Token management](#)
- [AmigoCloud API Browser](#)

5.3 ESRI ArcObjects

Driver short nameAO

Build dependenciesESRI ArcObjects

5.3.1 Overview

The OGR ArcObjects driver provides read-only access to ArcObjects based datasources. Since it uses the ESRI SDK, it has the requirement of needing an ESRI license to run. Nevertheless, this also means that the driver has full knowledge of ESRI abstractions. Among these, you have:

- GeoDatabases:
 - Personal GeoDatabase (.mdb)
 - File GeoDatabase (.gdb)
 - Enterprise GeoDatabase (.sde).
- ESRI Shapefiles

Although it has not been extended to do this yet (there hasn't been a need), it can potentially also support the following GeoDatabase Abstractions

- Annotation and Dimension feature classes
- Relationship Classes
- Networks (GN and ND)
- Topologies
- Terrains
- Representations
- Parcel Fabrics

You can try those above and they may work - but they have not been tested. Note the abstractions above cannot be supported with the Open FileGeoDatabase API.

5.3.2 Requirements

- An ArcView license or ArcEngine license (or higher) - Required to run.
- The ESRI libraries installed. This typically happens if you have ArcEngine or ArcGIS Desktop or Server installed - Required to compile. Note that this code should also compile using the ArcEngine *nix SDKs, however I do not have access to these and thus I have not tried it myself

5.3.3 Usage

Prefix the Datasource with “AO:”

Read from FileGDB and load into PostGIS:

```
ogr2ogr -overwrite -skipfailures -f "PostgreSQL" PG:"host=myhost user=myuser_
↳ dbname=mydb password=mypass" AO:"C:\somefolder\BigFileGDB.gdb" "MyFeatureClass"
```

Get detailed info of Personal GeoDatabase:

```
ogrinfo -al AO:"C:\somefolder\PersonalGDB.mdb"
```

Get detailed info of Enterprise GeoDatabase (.sde contains target version to connect to):

```
ogrinfo -al AO:"C:\somefolder\MySDEConnection.sde"
```

5.3.4 Building Notes

Read the [GDAL Windows Building example for Plugins](#). You will find a similar section in `nmake.opt` for ArcObjects. After you are done, go to the `$gdal_source_root/ogr/ogrsf_frmts/arcobjects*` folder and execute:

```
nmake /f makefile.vc plugin
nmake /f makefile.vc plugin-install
```

5.3.5 Known Issues

Date and blob fields have not been implemented. It is probably just a few lines of code, I just have not had time (or need) to do it.

5.4 ARCGEN - Arc/Info Generate

Driver short name

ARCGEN

Driver built-in by default

This driver is built-in by default

This driver reads files in Arc/Info Generate format. Those files are simple ASCII files that contain points, lines or polygons (one type of geometry per file).

5.4.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

5.4.2 See Also

- [Description of Arc/Info Generate format](#)

5.5 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.

5.5.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

5.5.2 See Also

- [AVCE00 Library Page](#)
- [AVCE00 OGR Driver \(.E00\)](#)

5.6 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.

5.6.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

5.6.2 See Also

- [AVCE00 Library Page](#)
- [AVCBin OGR Driver \(Binary Coverage\)](#)

5.7 BNA - Atlas BNA

Driver short name

BNA

Driver built-in by default

This driver is built-in by default

The BNA format is an ASCII exchange format for 2D vector data supported by many software packages. It only contains geometry and a few identifiers per record. Attributes must be stored into external files. It does not support any coordinate system information.

OGR has support for BNA reading and writing.

The OGR driver supports reading and writing of all the BNA feature types:

- points
- polygons
- lines
- ellipses/circles

As the BNA format is lacking from a formal specification, there can be various forms of BNA data files. The OGR driver does its best to parse BNA datasets and supports single line or multi line record formats, records with 2, 3 or 4 identifiers, etc etc. If you have a BNA data file that cannot be parsed correctly by the BNA driver, please report on the GDAL track system.

To be recognized as BNA, the file extension must be “.bna”. When reading a BNA file, the driver will scan it entirely to find out which layers are available. If the file name is foo.bna, the layers will be named foo_points, foo_polygons, foo_lines and foo_ellipses.

The BNA driver supports reading of polygons with holes or lakes. It determines what is a hole or a lake only from geometrical analysis (inclusion, non-intersection tests) and ignores completely the notion of polygon winding (whether the polygon edges are described clockwise or counter-clockwise). GDAL must be built with GEOS enabled to make geometry test work. Polygons are exposed as multipolygons in the OGR Simple Feature model.

Ellipses and circles are discretized as polygons with 360 points.

5.7.1 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.*)

5.7.2 Creation Issues

On export all layers are written to a single BNA file. Update of existing files is not currently supported.

If the output file already exists, the writing will not occur. You have to delete the existing file first.

The BNA writer supports the following creation options (dataset options):

- **LINEFORMAT**: By default when creating new .BNA 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).
- **MULTILINE**: By default, BNA files are created in the multi-line format (for each record, the first line contains the identifiers and the type/number of coordinates to follow. The following lines contain a pair of coordinates). This may be overridden through use of **MULTILINE=NO**.

- **NB_IDS**: BNA records may contain from 2 to 4 identifiers per record. Some software packages only support a precise number of identifiers. You can override the default value (2) by a precise value : **2**, **3** or **4**, or **NB_SOURCE_FIELDS**. **NB_SOURCE_FIELDS** means that the output file will contains the same number of identifiers as the features written in (clamped between 2 and 4).
- **ELLIPSES_AS_ELLIPSES**: the BNA writer will try to recognize ellipses and circles when writing a polygon. This will only work if the feature has previously been read from a BNA file. As some software packages do not support ellipses/circles in BNA data file, it may be useful to tell the writer by specifying **ELLIPSES_AS_ELLIPSES=NO** not to export them as such, but keep them as polygons.
- **NB_PAIRS_PER_LINE**: this option may be used to limit the number of coordinate pairs per line in multiline format.
- **COORDINATE_PRECISION**: this option may be used to set the number of decimal for coordinates. Default value is 10.

5.7.3 VSI Virtual File System API support

New in version 1.9.0: Some features below might require OGR >= 1.9.0

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.

5.7.4 Example

The `ogrinfo` utility can be used to dump the content of a BNA datafile :

```
ogrinfo -ro -al a_bna_file.bna
```

The `ogr2ogr` utility can be used to do BNA to BNA translation :

```
ogr2ogr -f BNA -dsco "NB_IDS=2" -dsco "ELLIPSES_AS_ELLIPSES=NO" output.bna input.bna
```

5.7.5 See Also

- [Description of the BNA file format](#)
- [Another description of the BNA file format](#)
- [Archive of Census Related Products \(ACRP\)](#) : downloadable BNA datasets of boundary files based on TIGER 1992 files containing U.S. census geographies

5.8 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.

5.8.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

5.8.2 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.

5.8.3 See Also

- [ODA DWG Reference](#)
- [Libopencad repository](#)

5.9 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.

5.9.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

5.9.2 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.

5.9.3 Configuration options

The following configuration options are available :

- `CARTO_API_URL`: defaults to `https://{[]account_name{[]}.carto.com/api/v2/sql`. Can be used to point to another server.
- `CARTO_HTTPS`: can be set to `NO` to use `http://` protocol instead of `https://` (only if `CARTO_API_URL` is not defined).
- `CARTO_API_KEY`: see following paragraph.

5.9.4 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.

5.9.5 Geometry

The OGR driver will report as many geometry fields as available in the layer (except the `'the_geom_webmercator'` field), following RFC 41.

5.9.6 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 `SetAttributeFilter()`.

5.9.7 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.

5.9.8 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 :

- `OGRFeature::CreateFeature()` \iff INSERT operation
- `OGRFeature::SetFeature()` \iff UPDATE operation
- `OGRFeature::DeleteFeature()` \iff DELETE operation
- `OGRDataSource::CreateLayer()` \iff CREATE TABLE operation
- `OGRDataSource::DeleteLayer()` \iff DROP TABLE operation

When inserting a new feature with `OGRFeature::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 `OGRFeature::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 `OGRDataSource::DeleteLayer()` and `OGRDataSource::CreateLayer()` to overwrite a table. Instead only call `OGRDataSource::CreateLayer()` with `OVERWRITE=YES`. This will avoid CARTO deleting maps that depend on this table

5.9.9 SQL

SQL commands provided to the `OGRDataSource::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.

5.9.10 Open options

Starting with GDAL 2.0, the following open options are available:

- **BATCH_INSERT=**YES/NO: Whether to group feature insertions in a batch. Defaults to YES. Only apply in creation or update mode.
- **COPY_MODE=**YES/NO: Using COPY for insertions and reads can result in a performance improvement. Defaults to YES.

5.9.11 Layer creation options

The following layer creation options are available:

- **OVERWRITE=**YES/NO: Whether to overwrite an existing table with the layer name to be created. Defaults to NO.
- **GEOMETRY_NULLABLE=**YES/NO: Whether the values of the geometry column can be NULL. Defaults to YES.
- **CARTODBFY=**YES/NO: Whether the created layer should be “Cartodbified” (i.e. registered in dashboard). Defaults to YES. 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: 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. The default value is “YES”. If enabled the table (layer) name will also be laundered.

5.9.12 Examples

Accessing data from a public table:

```
ogrinfo -ro "Carto:gdalautotest2 tables=tm_world_borders_simpl_0_3"
```

Creating and populating a table from a shapefile:

```
ogr2ogr --config CARTO_API_KEY abcdefghijklmnopqrstuvw -f Carto "Carto:myaccount" ↵  
↵myshapefile.shp
```

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.

5.9.13 See Also

- [Carto API overview](#)

5.10 Cloudant – Cloudant

Driver short name

Cloudant

Build dependencies

libcurl

Cloudant and CouchDB are API compatible and based on the same core technology. The geospatial extension for Cloudant is separate to GeoCouch. This driver can connect to the a Cloudant service, potentially enabled with the Cloudant spatial extension.

GDAL/OGR must be built with Curl support in order to the Cloudant driver to be compiled.

The driver supports read and write operations.

5.10.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

5.10.2 Cloudant vs OGR concepts

A Cloudant database is considered as a OGR layer. A Cloudant document is considered as a OGR feature. OGR preferably handles Cloudant documents following the GeoJSON specification.

5.10.3 Dataset name syntax

The syntax to open a Cloudant datasource is :

```
cloudant:http://example.com[/layername]
```

where <http://example.com> points to the root of a CouchDB repository and, optionally, `layername` is the name of a CouchDB database.

It is also possible to directly open a view :

```
cloudant:http://example.com/layername/_design/adesigndoc/_view/aview[?include_  
↪docs=true]
```

The `include_docs=true` might be needed depending on the value returned by the `emit()` call in the `map()` function.

5.10.4 Authentication

Some operations, in particular write operations, require authentication. The authentication can be passed with the `CLOUDANT_USERPWD` environment variable set to `user:password` or directly in the URL.

5.10.5 Filtering

The driver will forward any spatial filter set with `SetSpatialFilter()` to the server when the Cloudant extension is available.

By default, the driver will try the following spatial filter function “`_design/SpatialView/_geo/spatial`”, which is the valid spatial filter function for layers created by OGR. If that filter function does not exist, but another one exists, you can specify it with the `CLOUDANT_SPATIAL_FILTER` configuration option.

5.10.6 Paging

Features are retrieved from the server by chunks of 200 by default. Cloudant uses bookmarks to page through the data.

5.10.7 Write support

Table 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 `_rev` and use them.

5.10.8 Write support and OGR transactions

The `CreateFeature()/SetFeature()` 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 the `CreateFeature()/SetFeature()` operations between `OGRLayer::StartTransaction()` and `OGRLayer::CommitTransaction()`. The operations will be stored into memory and only executed at the time `CommitTransaction()` is called.

5.10.9 Layer creation options

The following layer creation options are supported:

- **UPDATE_PERMISSIONS** = LOGGED_USER|ALL|ADMIN|function(...)|DEFAULT : Update permissions for the new layer.
 - If set to LOGGED_USER (the default), only logged users will be able to make changes in the layer.
 - If set to ALL, all users will be able to make changes in the layer.
 - If set to ADMIN, only administrators will be able to make changes in the layer.
 - If beginning with “function(“, the value of the creation option will be used as the content of the [validate_doc_update](#) function.
 - Otherwise, all users will be allowed to make changes in non-design documents.
- **GEOJSON** = YES|NO : Set to NO to avoid writing documents as GeoJSON documents. Default to YES.
- **COORDINATE_PRECISION** = int_number : Maximum number of figures after decimal separator to write in coordinates. Default to 15. “Smart” truncation will occur to remove trailing zeros. Note: when opening a dataset in update mode, the `OGR_CLOUDANT_COORDINATE_PRECISION` configuration option can be set to have a similar role.

5.10.10 Examples

Listing the tables of a Cloudant repository:

```
ogrinfo -ro "cloudant:http://some_account.some_cloudant_server.com"
```

Creating and populating a table from a shapefile:

```
ogr2ogr -f cloudant "cloudant:http://some_account.some_cloudant_server.com" shapefile.  
↳ shp
```

5.10.11 See Also

- [CouchDB reference](#)
- [Cloudant Geospatial](#)
- [Documentation for ‘validate_doc_update’ function](#)

5.11 CouchDB - CouchDB/GeoCouch

Driver short name

CouchDB

Build dependencies

libcurl

This driver can connect to the a CouchDB service, potentially enabled with the GeoCouch spatial extension.

GDAL/OGR must be built with Curl support in order to the CouchDB driver to be compiled.

The driver supports read and write operations.

5.11.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

5.11.2 CouchDB vs OGR concepts

A CouchDB database is considered as a OGR layer. A CouchDB document is considered as a OGR feature.

OGR handles preferably CouchDB documents following the GeoJSON specification.

5.11.3 Dataset name syntax

The syntax to open a CouchDB datasource is :

```
couchdb:http://example.com[/layername]
```

where <http://example.com> points to the root of a CouchDB repository and, optionally, layername is the name of a CouchDB database.

It is also possible to directly open a view :

```
couchdb:http://example.com/layername/_design/adesigndoc/_view/aview[?include_
↪docs=true]
```

The `include_docs=true` might be needed depending on the value returned by the `emit()` call in the `map()` function.

5.11.4 Authentication

Some operations, in particular write operations, require authentication. The authentication can be passed with the `COUCHDB_USERPWD` environment variable set to `user:password` or directly in the URL.

5.11.5 Filtering

The driver will forward any spatial filter set with `SetSpatialFilter()` to the server when GeoCouch extension is available. It also makes the same for (very simple) attribute filters set with `SetAttributeFilter()`. When server-side filtering fails, it will default back to client-side filtering.

By default, the driver will try the following spatial filter function “`_design/ogr_spatial/_spatial/spatial`”, which is the valid spatial filter function for layers created by OGR. If that filter function does not exist, but another one exists, you can specify it with the `COUCHDB_SPATIAL_FILTER` configuration option.

Note that the first time an attribute request is issued, it might require write permissions in the database to create a new index view.

5.11.6 Paging

Features are retrieved from the server by chunks of 500 by default. This number can be altered with the `COUCHDB_PAGE_SIZE` configuration option.

5.11.7 Write support

Table 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 `_rev` and use them.

5.11.8 Write support and OGR transactions

The `CreateFeature()/SetFeature()` 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 the `CreateFeature()/SetFeature()` operations between `OGRLayer::StartTransaction()` and `OGRLayer::CommitTransaction()`. The operations will be stored into memory and only executed at the time `CommitTransaction()` is called.

5.11.9 Layer creation options

The following layer creation options are supported:

- **UPDATE_PERMISSIONS** = LOGGED_USER|ALL|ADMIN|function(...)DEFAULT : Update permissions for the new layer.
 - If set to LOGGED_USER (the default), only logged users will be able to make changes in the layer.
 - If set to ALL, all users will be able to make changes in the layer.
 - If set to ADMIN, only administrators will be able to make changes in the layer.
 - If beginning with “function(“, the value of the creation option will be used as the content of the `validate_doc_update function`.
 - Otherwise, all users will be allowed to make changes in non-design documents.
- **GEOJSON** = YES|NO : Set to NO to avoid writing documents as GeoJSON documents. Default to YES.
- **COORDINATE_PRECISION** = int_number : Maximum number of figures after decimal separator to write in coordinates. Default to 15. “Smart” truncation will occur to remove trailing zeros. Note: when opening a dataset in update mode, the `OGR_COUCHDB_COORDINATE_PRECISION` configuration option can be set to have a similar role.

5.11.10 Examples

Listing the tables of a CouchDB repository:

```
ogrinfo -ro "couchdb:http://some_account.some_couchdb_server.com"
```

Creating and populating a table from a shapefile:

```
ogr2ogr -f couchdb "couchdb:http://some_account.some_couchdb_server.com" shapefile.shp
```

5.11.11 See Also

- [CouchDB reference](#)
- [GeoCouch source code repository](#)
- [Documentation for 'validate_doc_update' function](#)

5.12 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.

While in theory .csv files could have any extension, in order to auto-recognise the format OGR only supports CSV files ending with the extension “.csv”. 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 1.8.0, for files structured as CSV, but not ending with .CSV extension, the ‘CSV:’ prefix can be added before the filename to force loading by the CSV driver.

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), Date-Time (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. Starting with GDAL 2.0, subtypes can be passed between parenthesis, such as “Integer(Boolean)”, “Integer(Int16)” and “Real(Float32)”. Starting with GDAL 2.1, 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

Starting with GDAL 2.2, the “JسونStringList”, “JسونIntegerList”, “JسونInteger64List” and “JسونRealList” 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 Jسون arrays, with proper CSV escaping.

Starting with GDAL 2.0, automatic field type guessing can also be done if specifying the open options described in the below “Open options” section.

5.12.1 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.)*

5.12.2 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 or a space (GDAL >= 2.0 for the later) character as field separator. This autodetection will work only if there's no other potential separator on the first line of the CSV file. Otherwise it will default to comma as separator.

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. Starting with GDAL 1.9.0 numeric values are treated as field names if they are enclosed in double quotes. Starting with GDAL 2.1, this behaviour can be modified via the HEADERS open option.

All CSV files are treated as UTF-8 encoded. Starting with GDAL 1.9.0, a Byte Order Mark (BOM) at the beginning of the file will be parsed correctly. From 1.9.2, The 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
132,55000.0,John Walker,"The "big" cheese."
133,11000.0,Jane Lake,Cleaning Staff
```

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.

5.12.3 Reading CSV containing spatial information

5.12.3.1 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"
```

Starting with GDAL 2.1, 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 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) 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
```

(continues on next page)

(continued from previous page)

```

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)

```

5.12.3.2 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)

```

5.12.4 Open options

Starting with GDAL 2.0, the following open options can be specified (typically with the -oo name=value parameters of ogrinfo or ogr2ogr):

- **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 (default 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 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**=size to specify the number of bytes to inspect to determine the data type and width/precision. The default will be 100000. Setting 0 means inspecting the whole file. Note : specifying a value over 1 MB (or 0 if the file is larger than 1MB) will prevent reading from standard input.
- **QUOTED_FIELDS_AS_STRING**=YES/NO (default 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. (GDAL >= 2.1) 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. (GDAL >= 2.1) 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. (GDAL >= 2.1) 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. (GDAL >= 2.1) 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 (default YES) Expose the detected X,Y,Z or geometry columns as regular attribute fields.
- **HEADERS**=YES/NO/AUTO (default AUTO) (GDAL >= 2.1) 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 (default NO) (GDAL >= 2.1) Whether to consider empty strings as null fields on reading'.

5.12.5 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. Starting with GDAL 2.1, 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.

Layer Creation options:

- **LINEFORMAT**: 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** (Starting with GDAL 1.6.0): 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** (Starting with GDAL 1.7.0): Create the associated .csvt file (see above paragraph) to describe the type of each column of the layer and its optional width and precision. Default value : NO
- **SEPARATOR=COMMA/SEMICOLON/TAB/SPACE** (Starting with GDAL 1.7.0): Field separator character. Default value : COMMA
- **WRITE_BOM=YES/NO** (Starting with GDAL 1.9.2): Write a UTF-8 Byte Order Mark (BOM) at the start of the file. Default value : NO
- **GEOMETRY_NAME=name** (Starting with GDAL 2.1): Name of geometry column. Only used if **GEOMETRY=AS_WKT** and **CREATE_CSVT=YES**. Defaults to WKT
- **STRING_QUOTING=IF_NEEDED/IF_AMBIGUOUS/ALWAYS** (Starting with GDAL 2.3): 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** (behaviour in older versions was **IF_NEEDED**)

Configuration options (set with “--config key value” on command line utilities):

- **OGR_WKT_PRECISION=int**: Number of decimals for coordinate values. Default to 15. A heuristic is used to remove insignificant trailing 00000x or 99999x that can appear when formatting decimal numbers.
- **OGR_WKT_ROUND=YES/NO**: (GDAL >=2.3) Whether to enable the above mentioned heuristics to remove insignificant trailing 00000x or 99999x. Default to YES.

5.12.6 VSI Virtual File System API support

(Some features below might require OGR >= 1.9.0)

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) , /vsizip/ (read-only) , /vsicurl/ (read-only) domains.

Writing to /dev/stdout or /vsistdout/ is also supported.

5.12.6.1 Examples

- This example shows using ogr2ogr to transform a shapefile with point geometry into a .csv file with the X,Y,Z coordinates of the points as first columns in the .csv file

```
ogr2ogr -f CSV output.csv input.shp -lco GEOMETRY=AS_XYZ
```

- This example shows using ogr2ogr to transform a shapefile into a .csv file with geography field formatted using GeoJSON format.

```
ogr2ogr -f CSV -dialect sqlite -sql "select AsGeoJSON(geometry) AS geom, * from ↵
↵input" output.csv input.shp
```

- Convert a CSV into a GeoPackage. Specify the names of the coordinate columns and assign a coordinate reference system.

```
ogr2ogr \
  -f GPKG output.gpkg \
  input.csv \
  -oo X_POSSIBLE_NAMES=longitude \
  -oo Y_POSSIBLE_NAMES=latitude \
  -a_srs 'EPSG:4326'
```

5.12.7 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 the [FAA website](#) (OGR >= 1.8.0)
- Files from the [USGS GNIS](#) (Geographic Names Information System) (OGR >= 1.9.0)
- The allCountries file from [GeoNames](#) (OGR >= 1.9.0 for direct import)
- [Eurostat .TSV files](#) (OGR >= 1.10.0)

5.12.8 Other Notes

- [GeoCSV specification](#) (supported by GDAL >= 2.1)
- 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.

5.13 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.

5.13.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

5.13.2 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`

The following open options are available:

- **URL:** URL to the CSW server endpoint (if not specified in the connection string already)
- **ELEMENTSETNAME=**brief/summary/full: Level of details of properties. Defaults to *full*.
- **FULL_EXTENT_RECORDS_AS_NON_SPATIAL=**YES/NO: Whether records with (-180,-90,180,90) extent should be considered non-spatial. Defaults to NO.
- **OUTPUT_SCHEMA=**URL : Value of outputSchema parameter, in the restricted set supported by the server. 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 : Maximum number of records to retrieve in a single time. Defaults to 500. Servers might have a lower accepted value.

5.13.3 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.

5.13.4 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.

5.13.5 Examples

Listing all the records of a CSW server:

```
ogrinfo -ro -al -noextent CSW:http://catalog.data.gov/csw
```

Listing all the records of a CSW server with spatial and an attribute filter on a give field:

```
ogrinfo -ro -al -noextent CSW:http://catalog.data.gov/csw -spat 2 49 2 49 -where
↳ "subject LIKE '%mineralogy%'"
```

Listing all the records of a CSW server that matches 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%'"
```

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
```

5.13.6 See Also

- [OGC CSW Standard](#)
- [GML driver documentation](#)

5.14 DB2 Spatial

Driver short name

DB2

Build dependencies

ODBC library

This driver implements support for access to spatial tables in the IBM DB2 for Linux, Unix and Windows (DB2 LUW) and the IBM DB2 for z/OS relational databases using the default ODBC support incorporated into GDAL.

The documentation for the DB2 spatial features can be found online for [DB2 for z/OS](#) and [DB2 LUW](#)

This driver is currently supported only in the Windows environment.

5.14.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

5.14.2 Connecting to a database

To connect to a DB2 datasource, use a connection string specifying the database name, with additional parameters as necessary. The connection strings must be prefixed with `'DB2ODBC:'`.

You can specify either a DSN that has been registered or use parameters that specify the host, port and protocol.

`DB2ODBC:database=dbname;DSN=datasourcename`

or

`DB2ODBC:database=dbname;DRIVER={IBM DB2 ODBC DRIVER};Hostname=hostipaddr;
↪PROTOCOL=TCPIP;port=db2port;UID=myuserid;PWD=mypw`

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 `DB2GSE.ST_GEOMETRY_COLUMNS` metadata view. You can omit specifying the schema and the geometry column portions of the syntax.

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.

5.14.3 Layers

By default the DB2 driver will only look for layers that are registered in the `DB2GSE.ST_GEOMETRY_COLUMNS` metadata table.

5.14.4 SQL statements

The DB2 driver passes SQL statements directly to DB2 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 DB2. It's also possible to request the OGR DB2 driver to handle SQL commands with the *OGR SQL* engine, by passing **"OGSQL"** string to the `ExecuteSQL()` method, as the name of the SQL dialect.

The DB2 driver in OGR supports the `OGRLayer::StartTransaction()`, `OGRLayer::CommitTransaction()` and `OGRLayer::RollbackTransaction()` calls in the normal SQL sense.

5.14.5 Creation Issues

This driver doesn't support creating new databases. Use the DB2 command line or tools like IBM Data Studio to create the database. It does allow creation of new layers within an existing database.

5.14.5.1 Layer Creation Options

- **OVERWRITE**: This may be "YES" to force an existing layer of the desired name to be destroyed before creating the requested layer.
- **LAUNDER**: This may be "YES" to force new fields created on this layer to have their field names "laundered" into a form more compatible with DB2. This converts to lower case and converts some special characters like "-" and "#" to "_". If "NO" exact names are preserved. The default value is "YES". If enabled the table (layer) name will also be laundered.
- **PRECISION**: 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. The default is "YES".
- **DIM={2,3}**: Control the dimension of the layer. Defaults to 2.
- **GEOM_NAME**: Set the name of geometry column in the new table. If omitted it defaults to *ogr_geometry*.
- **SCHEMA**: Set name of schema for new table. The default schema is that of the userid used to connect to the database
- **SRID**: 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.

5.14.5.2 Spatial Index Creation

By default the DB2 driver doesn't add spatial indexes to the tables during the layer creation. Spatial indexes should be created using the DB2 CREATE INDEX command.

5.14.6 Examples

Creating a layer from an OGR data source

```
ogr2ogr -overwrite DB2ODBC:database=sample;DSN=sampDSN zipcodes.shp
```

Connecting to a layer and dump the contents

```
ogrinfo -al DB2ODBC:database=sample;DSN=sampDSN;tables=zipcodes
```

5.15 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.

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.

5.15.1 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.)*

5.15.2 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.

5.15.3 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.

5.15.4 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.

The dataset creation supports the following options:

- **3D=***YES* or *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*: Indicate whether the whole seed file should be copied. If not, only the first three elements (and potentially the color table) will be copied. Default is NO.
- **COPY_SEED_FILE_COLOR_TABLE=***YES/NO*: Indicates whether the color table should be copied from the seed file. By default this is NO.
- **MASTER_UNIT_NAME=***name*: Override the master unit name from the seed file with the provided one or two character unit name.
- **SUB_UNIT_NAME=***name*: Override the sub unit name from the seed file with the provided one or two character unit name.
- **SUB_UNITS_PER_MASTER_UNIT=***count*: Override the number of subunits per master unit. By default the seed file value is used.
- **UOR_PER_SUB_UNIT=***count*: 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.

-
- [Dgnlib Page](#)
 - [Feature Style Specification](#)
 - [DGNv8 driver](#) (using Teigha libraries)

5.16 Microstation DGN v8

New in version 2.2.

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 Teiga 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.

5.16.1 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.)*

5.16.2 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.

5.16.3 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.

5.16.4 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.

5.16.5 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.

The dataset creation supports the following options:

- **SEED=filename**: Specify the seed file to use.
- **COPY_SEED_FILE_COLOR_TABLE=YES/NO**: Indicates whether the color table should be copied from the seed file. Only taken into account if SEED is specified. By default this is NO.
- **COPY_SEED_FILE_MODEL=YES/NO**: 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. By default this is YES.
- **COPY_SEED_FILE_MODEL_CONTROL_ELEMENTS=YES/NO**: 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. By default this is YES.
- **APPLICATION=string**: 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=string**: Set Title field in header. If not specified, from the seed file.
- **SUBJECT=string**: Set Subject field in header. If not specified, from the seed file.

- **AUTHOR=string**: Set Author field in header. If not specified, from the seed file.
- **KEYWORDS=string**: Set Keywords field in header. If not specified, from the seed file.
- **TEMPLATE=string**: Set Template field in header. If not specified, from the seed file.
- **COMMENTS=string**: Set Comments field in header. If not specified, from the seed file.
- **LAST_SAVED_BY=string**: Set LastSavedBy field in header. If not specified, from the seed file.
- **REVISION_NUMBER=string**: Set RevisionNumber field in header. If not specified, from the seed file.
- **CATEGORY=string**: Set Category field in header. If not specified, from the seed file.
- **MANAGER=string**: Set Manager field in header. If not specified, from the seed file.
- **COMPANY=string**: Set Company field in header. If not specified, from the seed file.

The layer creation supports the following options:

- **DESCRIPTION=string**: Description associated with the layer. If not specified, from the seed file.
- **DIM=2/3**: Dimension (ie 2D vs 3D) of the layer. By default, 3, unless the model is reused from the seed file.

- *DGN (v7) driver*
- *Feature Style Specification*

5.17 DODS/OPeNDAP

Driver short name

DODS

Build dependencies

libdap

This driver implements read-only support for reading feature data from OPeNDAP (DODS) servers. It is optionally included in OGR if built with OPeNDAP support libraries.

When opening a database, its name should be specified in the form “DODS:url”. The URL may include a constraint expression as shown here. Note that it may be necessary to quote or otherwise protect DODS URLs on the commandline if they include question mark or ampersand characters as these often have special meaning to command shells.

```
DODS:http://dods.gso.uri.edu/dods-3.4/nph-dods/broad1999?&press=148
```

By default top level Sequence, Grid and Array objects will be translated into corresponding layers. Sequences are (by default) treated as point layers with the point geometries picked up from lat and lon variables if available. To provide more sophisticated translation of sequence, grid or array items into features it is necessary to provide additional information to OGR as DAS (dataset auxiliary information) either from the remote server, or locally via the AIS mechanism.

A DAS definition for an OGR layer might look something like:

```
Attributes {
    ogr_layer_info {
        string layer_name WaterQuality;
        string spatial_ref WGS84;
        string target_container Water_Quality;
        layer_extents {
            Float64 x_min -180;
            Float64 y_min -90;
            Float64 x_max 180;
            Float64 y_max 90;
        }
        x_field {
            string name YR;
            string scope dds;
        }
        y_field {
            string name JD;
            string scope dds;
        }
    }
}
```

5.17.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

5.17.2 Caveats

- No field widths are captured for attribute fields from DODS.
- Performance for repeated requests is dramatically improved by enabling DODS caching. Try setting `USE_CACHE=1` in your `~/dodsrc`.

5.17.3 See Also

- [OPeNDAP](#)

5.18 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 approximation of arcs, ellipses, circles and rounded polylines 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 four degrees, but may be overridden by setting the configuration variable OGR_ARC_STEPSIZE.

5.18.1 DWG_INLINE_BLOCKS

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, and BlockAngle.
- 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.

5.18.2 Building

Currently DWG building is somewhat adhoc. On linux the normal practice is to hand edit `gdal/ogr/ogrsf_frmts/dwg/GNUMakefile`, update paths, and then build the driver as a plugin using the “make plugin” target.

5.19 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.

DXF files are considered to have no georeferencing information through OGR.

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.
- ExtendedEntity (GDAL <= 2.2.x): The values of extended entity attributes all appended to form a single text field, where available.
- RawCodeValues (GDAL >= 2.3.0): Only available when the configuration option `DXF_INCLUDE_RAW_CODE_VALUES` is set to TRUE. A string list containing all group codes and values that are not handled by the DXF reader.
- 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.

5.19.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

5.19.2 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` to `FALSE`.
- 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.
- MLINE:
 - (GDAL \geq 2.3.0) 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.
 - (GDAL \leq 2.2.x) No support.
- CIRCLE, ELLIPSE, ARC, SPLINE, (GDAL \geq 2.3.0) HELIX: Translated as a LINESTRING, tessellating the curve into line segments. (GDAL \geq 2.3.0) 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: 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: See below.
- (GDAL \geq 2.3.0) DXF_FEATURE_LIMIT_PER_BLOCK: Maximum number of features inserted from a single block. Set to -1 for no limit. Defaults to 10000.
- ATTDEF, ATTRIB:
 - (GDAL \geq 2.3.0) 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).
 - (GDAL \leq 2.2.x) ATTDEF entities are treated as TEXT. ATTRIB entities are not supported.
- 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.

(GDAL \geq 2.3.0) The DXF_HATCH_TOLERANCE config option determines the tolerance used when looking for the next component to add to the hatch boundary.

(GDAL \leq 2.2.x) Only line and polyline boundary paths are translated correctly.

- 3DFACE, SOLID, (GDAL >= 2.3.0) 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:
 - (GDAL >= 2.3.0) 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.
 - (GDAL <= 2.2.x) Dimensions are translated as a MULTILINESTRING and a POINT for the text.
- LEADER, MULTILEADER:
 - (GDAL >= 2.3.0) 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.
 - (GDAL <= 2.2.x) No support.
- 3DSOLID, REGION, 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 approximation of arcs, ellipses, circles and rounded polylines as linestrings is done by splitting the arcs into subarcs of no more than a threshold angle. This angle is set using the OGR_ARC_STEPSIZE configuration option. This defaults to 4 degrees. You can also set the OGR_ARC_MAX_GAP configuration option to enforce a maximum distance between adjacent points on the interpolated curve. Setting this option to 0 (the default) means no maximum distance applies.

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.

5.19.3 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. (Note, in GDAL 2.2.x and earlier this field was called BlockName.)
- (GDAL >= 2.3.0) 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 (GDAL >= 2.3.0): The unit normal vector of the object coordinate system (OCS) of the INSERT entity.
- BlockOCSCoords (GDAL >= 2.3.0): The OCS coordinates of the insertion point.
- BlockAttributes (GDAL >= 2.3.0): 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.

5.19.4 3D Extensibility

DXF files may contain 3DSOLID, REGION and SURFACE entities, which contain 3D modelling data in the undocumented Autodesk ShapeModeler (ASM) format. GDAL cannot transform these entities into OGR geometries, so they are skipped by default.

Starting from GDAL 2.3.0, the DXF_3D_EXTENSIBLE_MODE configuration option may be set to TRUE to include these entities with the raw ASM data stored in a field. This option will add two new fields:

- ASMData: A binary field that contains the ASM data.
- ASMTransform: A 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.

5.19.5 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. In GDAL 1.8.x and earlier this was ignored but from GDAL 1.9.0 and later 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. It could be edited manually, or the DXF_ENCODING configuration variable can be used to override what id will be used by OGR in transcoding. The value of DXF_ENCODING should be an encoding name supported by CPLRecode() (i.e. an iconv name), not a DXF \$DWGCODEPAGE name. Using a DXF_ENCODING name of “UTF-8” will avoid any attempt to recode the text as it is read.

5.19.6 Creation Issues

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 to FALSE. 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)
SYMBOL	Point	color (c)
LABEL	Point	GDAL >= 2.3.0: 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) GDAL <= 2.2.x: text (t); font size (s), treated as cap height; text color (c); angle (a); anchor point (p)

The dataset creation 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.

Note that in GDAL 1.8 and later, 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).

5.19.6.1 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. (Note, in GDAL 2.2.x and earlier this attribute was called BlockName.)
- 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.

5.19.6.2 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”).

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.

Units

GDAL writes DXF files with measurement units set to “Imperial - Inches”. If you need to change the units, edit the `$MEASUREMENT` and `$INSUNITS` variables in the header template.

See also

[AutoCAD 2000 DXF Reference](#)

[AutoCAD 2014 DXF Reference](#)

[DXF header reference](#)

5.20 Google Earth Engine Data API

New in version 2.4.

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.

5.20.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

5.20.2 Dataset name syntax

The minimal syntax to open a datasource is :

`EEDA:[collection]`

where collection is something like projects/earthengine-public/assets/COPERNICUS/S2.

5.20.3 Open options

The following open options are available :

- **COLLECTION**=string: To specify the collection if not specified in the connection string.

5.20.4 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.
- 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` to YES.

5.20.5 Configuration options

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**=string: 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**=string: email to be specified together with `EEDA_PRIVATE_KEY/EEDA_PRIVATE_KEY_FILE` to use OAuth2 Service Account authentication.

5.20.6 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. EEDA1:projects/earthengine-public/assets/COPERNICUS/S2/20170430T190351_20170430T190351_T10SEG) that can be opened with the Google Earth Engine Data API Image driver	No
start-Time	Date-Time	Acquisition start date	Yes (restricted to \geq comparison on top level)
end-Time	Date-Time	Acquisition end date	Yes (restricted to \leq comparison on top level)
update-Time	Date-Time	Update date	No
size-Bytes	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).

5.20.6.1 Geometry

The footprint of each image is reported as a MultiPolygon with a longitude/latitude WGS84 coordinate system (EPSG:4326).

5.20.6.2 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.

5.20.6.3 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.

5.20.6.4 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.

5.20.6.5 Examples

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
```

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
```

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
```

5.20.7 See Also:

- *Google Earth Engine Data API Image driver*

5.21 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 Impôts - 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.4 resource files.

The whole set of files will be parsed into memory. This may be a limitation if dealing with big EDIGEO exchanges.

5.21.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (vsimem/, etc.)*

5.21.2 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_ATTR_VAL: the value of the attribute to display (found in the ATTR 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 that defaults to 2.

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.

5.21.3 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 \(in English\)](#)
- [Sample of EDIGEO data](#)

5.22 Elasticsearch: Geographically Encoded Objects for Elasticsearch

Driver short name

Elasticsearch

Build dependencies

libcurl

Driver is read-write starting with GDAL 2.1 (was write only in GDAL 2.0 or earlier)

As of GDAL 2.1, Elasticsearch 1.X and, partially, 2.X versions are supported (5.0 known not to work). GDAL 2.2 adds supports for Elasticsearch 2.X and 5.X

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.

5.22.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

5.22.2 Opening dataset name syntax

Starting with GDAL 2.1, 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

The open options available are :

- **HOST**=hostname: Server hostname. Default to localhost.
- **PORT**=port. Server port. Default to 9200.
- **USERPWD**=user:password. (GDAL >=2.4) Basic authentication as username:password.
- **LAYER**=name. (GDAL >=2.4) Index name or index_mapping to use for restricting layer listing.
- **BATCH_SIZE**=number. Number of features to retrieve per batch. Default is 100.
- **FEATURE_COUNT_TO_ESTABLISH_FEATURE_DEFN**=number. Number of features to retrieve to establish feature definition. -1 = unlimited. Defaults to 100.
- **JSON_FIELD**=YES/NO. Whether to include a field called “_json” with the full document as JSON. Defaults to NO.
- **FLATTEN_NESTED_ATTRIBUTE**=YES/NO. Whether to recursively explore nested objects and produce flatten OGR attributes. Defaults to YES.
- **FID**=string. Field name, with integer values, to use as FID. Defaults to ‘ogc_fid’
- **FORWARD_HTTP_HEADERS_FROM_ENV**=string. (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.

5.22.3 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.

5.22.4 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
```

The Elasticsearch writer supports the following Configuration Options. Starting with GDAL 2.1, layer creation options are also available and should be preferred:

- **ES_WRITEMAP**=/path/to/mapfile.txt. 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. Starting with GDAL 2.1, the **WRITE_MAPPING** layer creation option should rather be used.
- **ES_META**=/path/to/mapfile.txt. Tells the driver to the user-defined field mappings. Starting with GDAL 2.1, the **MAPPING** layer creation option should rather be used.
- **ES_BULK**=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. Starting with GDAL 2.1, the **BULK_SIZE** layer creation option should rather be used.
- **ES_OVERWRITE**=1. Overwrites the current index by deleting an existing one. Starting with GDAL 2.1, the **OVERWRITE** layer creation option should rather be used.

5.22.5 Geometry types

In GDAL 2.0 and earlier, the driver was limited in the geometry it handles: even if polygons were provided as input, they were stored as [geo point](#) and the “center” of the polygon is used as value of the point. Starting with GDAL 2.1, [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).

5.22.6 Filtering

The driver will forward any spatial filter set with `SetSpatialFilter()` to the server.

Starting with GDAL 2.2, 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.

5.22.7 Paging

Features are retrieved from the server by chunks of 100. This can be altered with the `BATCH_SIZE` open option.

5.22.8 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).

5.22.9 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.

5.22.10 ExecuteSQL() interface

Starting with GDAL 2.2, 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.

5.22.11 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.

5.22.12 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.

5.22.13 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.

5.22.14 Layer creation options

Starting with GDAL 2.1, the driver supports the following layer creation options:

- **INDEX_NAME**=name. Name of the index to create (or reuse). By default the index name is the layer name.
- **INDEX_DEFINITION**=filename or JSon. (GDAL >= 2.4) Filename from which to read a user-defined index definition, or inlined index definition as serialized JSon.
- **MAPPING_NAME**==name. (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 or 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. Whether to overwrite an existing type mapping with the layer name to be created. Defaults to NO.
- **OVERWRITE_INDEX**=YES/NO. (GDAL >= 2.2) Whether to overwrite the whole index to which the layer belongs to. Defaults to NO. 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**=name. Name of geometry column. Defaults to 'geometry'.
- **GEOM_MAPPING_TYPE**=AUTO/GEO_POINT/GEO_SHAPE. Mapping type for geometry fields. Defaults to AUTO. GEO_POINT uses the [geo_point](#) mapping type. If used, the “centroid” of the geometry is used. This is the behaviour of GDAL < 2.1. GEO_SHAPE uses the [geo_shape](#) mapping type, compatible of all geometry types. When using AUTO, for geometry fields of type Point, a [geo_point](#) is used. In other cases, [geo_shape](#) is used.
- **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. Whether fields should be stored in the index. Setting to YES sets the “[store](#)” [property](#) of the field mapping to “true” for all fields. Defaults to NO. (Note: prior to GDAL 2.1, the default behaviour was to store fields) This option is without effect if MAPPING is specified.
- **STORED_FIELDS**=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**=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. Starting with GDAL 2.2, the {ALL} value can be used to designate all fields. This option is without effect if MAPPING is specified.
- **NOT_INDEXED_FIELDS**=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**=(GDAL > 2.2) 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. Whether to use bulk insert for feature creation. Defaults to YES.
- **BULK_SIZE**=value. Size in bytes of the buffer for bulk upload. Defaults to 1000000 (1 million).
- **FID**=string. Field name, with integer values, to use as FID. Can be set to empty to disable the writing of the FID value. Defaults to 'ogc_fid'
- **DOT_AS_NESTED_FIELD**=YES/NO. Whether to consider dot character in field name as sub-document. Defaults to YES.

- **IGNORE_SOURCE_ID=YES/NO.** Whether to ignore `_id` field in features passed to `CreateFeature()`. Defaults to NO.

5.22.15 Examples

Open the local store:

```
ogrinfo ES:
```

Open a remote store:

```
ogrinfo ES:http://example.com:9200
```

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\" } } }"
```

Load an Elasticsearch index with a shapefile:

```
ogr2ogr -f "Elasticsearch" http://localhost:9200 my_shapefile.shp
```

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 --config ES_WRITE_MAP /path/to/file/map.txt -f "Elasticsearch" http://
↪ localhost:9200 my_shapefile.shp
```

or (GDAL >= 2.1):

```
ogr2ogr -progress -lco WRITE_MAPPING=/path/to/file/map.txt -f "Elasticsearch" http://
↪ localhost:9200 my_shapefile.shp
```

Read the Mapping File: Reads the mapping file during the transformation

```
ogr2ogr -progress --config ES_META /path/to/file/map.txt -f "Elasticsearch" http://
↪ localhost:9200 my_shapefile.shp
```

or (GDAL >= 2.1):

```
ogr2ogr -progress -lco MAPPING=/path/to/file/map.txt -f "Elasticsearch" http://
↪ localhost:9200 my_shapefile.shp
```

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 --config ES_BULK 5000000 -f "Elasticsearch" http://localhost:9200_
↳PG:"host=localhost user=postgres dbname=my_db password=password" "my_table" -nln_
↳thetable
```

or (GDAL >= 2.1):

```
ogr2ogr -progress -lco BULK_SIZE=5000000 -f "Elasticsearch" http://localhost:9200 my_
↳shapefile.shp
```

Overwrite the current Index: If specified, this will overwrite the current index. Otherwise, the data will be appended.

```
ogr2ogr -progress --config ES_OVERWRITE 1 -f "Elasticsearch" http://localhost:9200 PG:
↳"host=localhost user=postgres dbname=my_db password=password" "my_table" -nln_
↳thetable
```

or (GDAL >= 2.1):

```
ogr2ogr -progress -overwrite ES:http://localhost:9200 PG:"host=localhost_
↳user=postgres dbname=my_db password=password" "my_table" -nln thetable
```

5.22.16 See Also

- [Home page for Elasticsearch](#)
- [Examples Wiki](#)
- [Google Group](#)

5.23 ESRIJSON / FeatureService driver

Driver short name

ESRIJSON

Driver built-in by default

This driver is built-in by default

(Note: prior to GDAL 2.3, the functionality of this driver was available in the GeoJSON driver. They are now distinct drivers)

This driver can read the JSON output of Feature Service requests following the [GeoServices REST Specification](#), like implemented by [ArcGIS Server REST API](#). Starting with OGR 2.0, 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.

5.23.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

5.23.2 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

Starting with GDAL 2.3, the URL/filename/text might be prefixed with ESRIJSON: to avoid any ambiguity with other drivers.

5.23.3 Open options

(GDAL >= 2.0)

- **FEATURE_SERVER_PAGING** = YES/NO: Whether to automatically scroll through results with a ArcGIS Feature Service endpoint.

5.23.4 Example

Read the result of a FeatureService request against a GeoServices REST server:

```
ogrinfo -ro -al "http://sampleserver3.arcgisonline.com/ArcGIS/rest/services/
↳Hydrography/Watershed173811/FeatureServer/0/query?where=objectid+%3D+objectid&
↳outfields=*&f=json"
```

5.23.5 See Also

- *GeoJSON driver*
- GeoServices REST Specification

5.24 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 : starting with OGR 1.11, the *OpenFileGDB driver* exists as an alternative built-in i.e. not depending on a third-party library) read-only driver.

5.24.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

5.24.2 Requirements

- FileGDB API SDK
- OGR >= 1.9.0

Curve in geometries are supported on reading with GDAL >= 2.2.

5.24.3 Bulk feature loading (OGR >= 1.9.2)

The `FGDB_BULK_LOAD` configuration option can be set to YES to speed-up feature insertion (or sometimes solve problems when inserting a lot of features (see <http://trac.osgeo.org/gdal/ticket/4420>). The effect of this configuration option is to cause a write lock to be taken and a temporary disabling of the indexes. Those are restored when the datasource is closed or when a read operation is done.

Starting with GDAL 2.0, bulk load is enabled by default for newly created layers (unless otherwise specified).

5.24.4 SQL support (OGR >= 1.10)

Starting with OGR 1.10, 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`.

5.24.4.1 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.

5.24.5 Transaction support (OGR >= 2.0)

The FileGDB driver implements transactions at the database level, through an emulation (as per rfc-54), since the FileGDB SDK itself does not offer it. This works by backing up the current state of a geodatabase when `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. So this might be costly when operating on huge geodatabases.

Starting with GDAL 2.1, on Linux/Unix, instead of a full backup copy only layers that are modified are backed up.

Note that this emulation has an unspecified behaviour in case of concurrent updates (with different connections in the same or another process).

5.24.6 CreateFeature() support

The FileGDB SDK API does not allow to create a feature with a FID specified by the user. Starting with GDAL 2.1, the FileGDB driver implements a special FID remapping technique to enable the user to create features at the FID of their choice.

5.24.7 Dataset Creation Options

None.

5.24.8 Layer Creation Options

- **FEATURE_DATASET**: 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=string**: (GDAL >=2.3) Set layer name alias.
- **GEOMETRY_NAME**: Set name of geometry column in new layer. Defaults to “SHAPE”.
- **GEOMETRY_NULLABLE**: (GDAL >=2.0) Whether the values of the geometry column can be NULL. Can be set to NO so that geometry is required. Default to “YES”
- **FID**: Name of the OID column to create. Defaults to “OBJECTID”. Note: option was called OID_NAME in releases before GDAL 2

- **XYTOLERANCE, ZTOLERANCE**: 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 be 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.

- **XORIGIN, YORIGIN, ZORIGIN, XYSCALE, ZSCALE**: 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.
- **XML_DEFINITION** : (GDAL >= 1.10) When this option is set, its value will be used as the XML definition to create the new table. The root node of such a XML definition must be a <esri:DataElement> element conformant to FileGDBAPI.xsd
- **CREATE_MULTIPATCH=YES** : (GDAL >= 1.11) When this option is set, geometries of layers of type MultiPolygon will be written as MultiPatch
- **CONFIGURATION_KEYWORD=DEFAULTS/TEXT_UTF16/MAX_FILE_SIZE_4GB/MAX_FILE_SIZE_256TB/GEOMETRY** : (GDAL >= 2.0) Customize how data is stored. By default text in UTF-8 and data up to 1TB

5.24.9 Examples

- Read layer from FileGDB and load into PostGIS:
- Get detailed info for FileGDB:

5.24.10 Building Notes

Read the [GDAL Windows Building example for Plugins](#). You will find a similar section in `nmake.opt` for FileGDB. After you are done, go to the `$gdal_source_root\ogr\ogr_srf_frmts\filegdb` folder and execute:

```
nmake /f makefile.vc plugin nmake /f makefile.vc plugin-install
```

5.24.11 Known Issues

- 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 XXXXXXXX. 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.
- Driver can’t read data in SDC format (Smart Data Compression) because operation is not supported by the ESRI SDK.
- Reading data compressed in CDF format (Compressed Data Format) requires ESRI SDK 1.4 or later.

5.24.12 Links

- [ESRI File Geodatabase API Page](#)
- *OpenFileGDB driver*, not depending on a third-party library/SDK

5.25 FlatGeobuf

New 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.

5.25.1 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.*)

5.25.2 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.

5.25.3 Open options

- **VERIFY_BUFFERS=YES/NO**: Set to YES to verify buffers when reading. This can provide some protection for invalid/corrupt data with a performance trade off. Defaults to YES.

5.25.4 Dataset Creation Options

None

5.25.5 Layer Creation Options

- **SPATIAL_INDEX=YES/NO**: Set to YES to create a spatial index. Defaults to YES.
- **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.

5.25.6 Examples

- Simple translation of a single 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
```

- Conversion of a Geopackage file with multiple layers:

```
% ogr2ogr -f FlatGeobuf my_fgb_dataset input.gpkg
```

5.25.7 See Also

- [FlatGeobuf at GitHub](#)

5.26 FMEObjects Gateway

Driver short name

FME

Build dependencies

FME

Feature sources supported by FMEObjects are supported for reading by OGR if the FMEObjects gateway is configured, and if a licensed copy of FMEObjects is installed and accessible.

To using the FMEObjects based readers the data source name passed should be the name of the FME reader to use, a colon and then the actual data source name (i.e. the filename). For instance, “NTF:F:DATANTF2144.NTF” would indicate the NTF reader should be used to read the file There are a number of special cases:

- A data source ending in .fdd will be assumed to be an “FME Datasource Definition” file which will contain the reader name, the data source name, and then a set of name/value pairs of lines for the macros suitable to pass to the createReader() call.
- A datasource named PROMPT will result in prompting the user for information using the regular FME dialogs. This only works on Windows.
- A datasource named “PROMPT:filename” will result in prompting, and then having the resulting definition saved to the indicate files in .fdd format. The .fdd extension will be forced on the filename. This only works on Windows.

Each FME feature type will be treated as a layer through OGR, named by the feature type. With some limitations FME coordinate systems are supported. All FME geometry types should be properly supported. FME graphical attributes (color, line width, etc) are not converted into OGR Feature Style information.

5.26.1 Caching

In order to enable fast access to large datasets without having to retranslate them each time they are accessed, the FMEObjects gateway supports a mechanism to cache features read from FME readers in “Fast Feature Stores”, a native vector format for FME with a spatial index for fast spatial searches. These cached files are kept in the directory indicated by the OGRFME_TMPDIR environment variable (or TMPDIR or /tmp or C:\ if that is not available).

The cached feature files will have the prefix FME_OLEDB_ and a master index is kept in the file ogrfmeds.ind. To clear away the index delete all these files. Do not just delete some.

By default features in the cache are re-read after 3600s (60 minutes). Cache retention times can be altered at compile time by altering the fme2ogr.h include file.

Input from the SDE and ORACLE readers are not cached. These sources are treated specially in a number of other ways as well.

5.26.2 Caveats

1. Establishing an FME session is quite an expensive operation, on a 350Mhz Linux system this can be in excess of 10s.
2. Old files in the feature cache are cleaned up, but only on subsequent visits to the FMEObjects gateway code in OGR. This means that if unused the FMEObjects gateway will leave old cached features around indefinitely.

5.26.3 Build/Configuration

To include the FMEObjects gateway in an OGR build it is necessary to have FME loaded on the system. The *–with-fme=\$FME_HOME* configuration switch should be supplied to configure. The FMEObjects gateway is not explicitly linked against (it is loaded later when it may be needed) so it is practical to distribute an OGR binary build with FMEObjects support without distributing FMEObjects. It will just “work” for people who have FMEObjects in the path.

The FMEObjects gateway has been tested on Linux and Windows.

More information on the FME product line, and how to purchase a license for the FME software (enabling FMEObjects support) can be found on the Safe Software web site at www.safe.com. Development of this driver was financially supported by Safe Software.

5.27 GeoConcept text export

Driver short name

Geoconcept

Driver built-in by default

This driver is built-in by default

GeoConcept text export files should be available for writing and reading.

The OGR GeoConcept driver treats a single GeoConcept file within a directory as a dataset comprising layers. GeoConcept files extensions are `.txt` or `.gxt`.

Currently the GeoConcept driver only supports multi-polygons, lines and points.

5.27.1 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.)*

5.27.2 GeoConcept Text File Format (gxt)

GeoConcept is a GIS developed by the Company GeoConcept SA.

It's an object oriented GIS, where the features are named « objects », and feature types are named « type/subtype » (class allowing inheritance).

Among its import/export formats, it proposes a simple text format named gxt. A gxt file may contain objects from several type/subtype.

GeoConcept text export files should be available for writing and reading.

The OGR GeoConcept driver treats a single GeoConcept file within a directory as a dataset comprising layers. GeoConcept files extensions are `.txt` or `.gxt`.

Currently the GeoConcept driver only supports multi-polygons, lines and points.

5.27.3 Creation Issues

The GeoConcept driver treats a GeoConcept file (`.txt` or `.gxt`) as a dataset.

GeoConcept files can store multiple kinds of geometry (one by layer), even if a GeoConcept layer can only have one kind of geometry.

Note this makes it very difficult to translate a mixed geometry layer from another format into GeoConcept format using `ogr2ogr`, since `ogr2ogr` has no support for separating out geometries from a source layer.

GeoConcept sub-type is treated as OGR feature. The name of a layer is therefore the concatenation of the GeoConcept type name, `'.'` and GeoConcept sub-type name.

GeoConcept type definition (`.gct` files) are used for creation only.

GeoConcept feature fields definition are stored in an associated `.gct` file, and so fields suffer a number of limitations (FIXME) :

- Attribute names are not limited in length.
- Only Integer, Real and String field types are supported. The various list, and other field types cannot be created for the moment (they exist in the GeoConcept model, but are not yet supported by the GeoConcept driver).

The OGR GeoConcept driver does not support deleting features.

5.27.3.1 Dataset Creation Options

EXTENSION=TXT|GXT : indicates the GeoConcept export file extension. TXT was used by earlier releases of GeoConcept. GXT is currently used.

CONFIG=path to the GCT : the GCT file describe the GeoConcept types definitions : In this file, every line must start with `//#` followed by a keyword. Lines starting with `//` are comments.

It is important to note that a GeoConcept export file can hold different types and associated sub-types.

- configuration section : the GCT file starts with `//#SECTION CONFIG` and ends with `//#ENDSECTION CONFIG`. All the configuration is enclosed within these marks.
- map section : purely for documentation at the time of writing this document. This section starts with `//#SECTION MAP` and ends with `//#ENDSECTION MAP`.
- type section : this section defines a class of features. A type has a name (keyword `Name`) and an ID (keyword `ID`). A type holds sub-types and fields. This section starts with `//#SECTION TYPE` and ends with `//#ENDSECTION TYPE`.
 - sub-type section : this sub-section defines a kind of features within a class. A sub-type has a name (keyword `Name`), an ID (keyword `ID`), a type of geometry (keyword `Kind`) and a dimension. The following types of geometry are supported : POINT, LINE, POLYGON. The current release of this driver does not support the TEXT geometry. The dimension can be 2D, 3DM or 3D. A sub-type holds fields. This section starts with `//#SECTION SUBTYPE` and ends with `//#ENDSECTION SUBTYPE`.
 - * fields section : defines user fields. A field has a name (keyword `Name`), an ID (keyword `ID`), a type (keyword `Kind`). The following types of fields are supported : INT, REAL, MEMO, CHOICE, DATE, TIME, LENGTH, AREA. This section starts with `//#SECTION FIELD` and ends with `//#ENDSECTION FIELD`.
 - field section : defines type fields. See above.
- field section : defines general fields. Out of these, the following rules apply :
 - private field names start with a '@' : the private fields are Identifier, Class, Subclass, Name, NbFields, X, Y, XP, YP, Graphics, Angle.
 - some private field are mandatory (they must appear in the configuration) : Identifier, Class, Subclass, Name, X, Y.
 - If the sub-type is linear (LINE), then the following fields must be declared XP, YP.
 - If the sub-type is linear or polygonal (LINE, POLY), then Graphics must be declared.
 - If the sub-type is ponctual or textual (POINT, TEXT), the Angle may be declared.

When this option is not used, the driver manage types and sub-types name based on either the layer name or on the use of `-nln` option.

5.27.3.2 Layer Creation Options

FEATURETYPE=TYPE.SUBTYPE : defines the feature to be created. The **TYPE** corresponds to one of the Name found in the GCT file for a type section. The **SUBTYPE** corresponds to one of the Name found in the GCT file for a sub-type section within the previous type section.

At the present moment, coordinates are written with 2 decimals for Cartesian spatial reference systems (including height) or with 9 decimals for geographical spatial reference systems.

5.27.3.3 Examples

Example of a .gct file :

```
// #SECTION CONFIG
// #SECTION MAP
// # Name=SCAN1000-TILES-LAMB93
// # Unit=m
// # Precision=1000
// #ENDSECTION MAP
// #SECTION TYPE
// # Name=TILE
// # ID=10
// #SECTION SUBTYPE
// # Name=TILE
// # ID=100
// # Kind=POLYGON
// # 3D=2D
// #SECTION FIELD
// # Name=IDSEL
// # ID=101
// # Kind=TEXT
// #ENDSECTION FIELD
// #SECTION FIELD
// # Name=NOM
// # ID=102
// # Kind=TEXT
// #ENDSECTION FIELD
// #SECTION FIELD
// # Name=WITHDATA
// # ID=103
// # Kind=INT
// #ENDSECTION FIELD
// #ENDSECTION SUBTYPE
// #ENDSECTION TYPE
// #SECTION FIELD
// # Name=@Identifier
// # ID=-1
// # Kind=INT
// #ENDSECTION FIELD
// #SECTION FIELD
// # Name=@Class
// # ID=-2
// # Kind=CHOICE
// #ENDSECTION FIELD
// #SECTION FIELD
// # Name=@Subclass
```

(continues on next page)

(continued from previous page)

```
// # ID=-3
// # Kind=CHOICE
// #ENDSECTION FIELD
// #SECTION FIELD
// # Name=@Name
// # ID=-4
// # Kind=TEXT
// #ENDSECTION FIELD
// #SECTION FIELD
// # Name=@X
// # ID=-5
// # Kind=REAL
// #ENDSECTION FIELD
// #SECTION FIELD
// # Name=@Y
// # ID=-6
// # Kind=REAL
// #ENDSECTION FIELD
// #SECTION FIELD
// # Name=@Graphics
// # ID=-7
// # Kind=REAL
// #ENDSECTION FIELD
// #ENDSECTION CONFIG
```

Example of a GeoConcept text export :

```
// $DELIMITER " "
// $QUOTED-TEXT "no"
// $CHARSET ANSI
// $UNIT Distance=m
// $FORMAT 2
// $SYSCOORD {Type: 2001}
// $FIELDS Class=TILE;Subclass=TILE;Kind=4;Fields=Private#Identifier Private#Class
↪ Private#Subclass Private#Name Private#NbFields IDSEL NOM WITHDATA
↪ Private#X Private#Y Private#Graphics
-1 TILE TILE TILE 3 000-2007-0050-7130-LAMB93 0 50000.00
↪ 7130000.00 4 600000.00 7130000.00 600000.00 6580000.00 50000.00
↪ 6580000.00 50000.00 7130000.00
-1 TILE TILE TILE 3 000-2007-0595-7130-LAMB93 0 595000.00
↪ 7130000.00 4 1145000.00 7130000.00 1145000.00 6580000.00 595000.
↪ 00 6580000.00 595000.00 7130000.00
-1 TILE TILE TILE 3 000-2007-0595-6585-LAMB93 0 595000.00
↪ 6585000.00 4 1145000.00 6585000.00 1145000.00 6035000.00 595000.
↪ 00 6035000.00 595000.00 6585000.00
-1 TILE TILE TILE 3 000-2007-1145-6250-LAMB93 0 1145000.00
↪ 6250000.00 4 1265000.00 6250000.00 1265000.00 6030000.00 1145000.
↪ 00 6030000.00 1145000.00 6250000.00
-1 TILE TILE TILE 3 000-2007-0050-6585-LAMB93 0 50000.00
↪ 6585000.00 4 600000.00 6585000.00 600000.00 6035000.00 50000.00
↪ 6035000.00 50000.00 6585000.00
```

Example of use :

Creating a GeoConcept export file :

```
ogr2ogr -f "Geoconcept" -a_srs "+init=IGNF:LAMB93" -dsco EXTENSION=txt -dsco_
↳CONFIG=tile_schema.gct tile.gxt tile.shp -lco FEATURETYPE=TILE.TILE
```

Appending new features to an existing GeoConcept export file :

```
ogr2ogr -f "Geoconcept" -update -append tile.gxt tile.shp -nln TILE.TILE
```

Translating a GeoConcept export file layer into MapInfo file :

```
ogr2ogr -f "MapInfo File" -dsco FORMAT=MIF tile.mif tile.gxt TILE.TILE
```

5.27.3.4 See Also

- [GeoConcept web site](#)

5.28 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](#): Data-source, Layer, Feature, Geometry. The implementation is based on [GeoJSON Specification, v1.0](#).

Starting with GDAL 2.1.0, the GeoJSON driver supports updating existing GeoJSON files. In that case, the default value for the `NATIVE_DATA` open option will be YES.

5.28.1 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.*)

5.28.2 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

Starting with GDAL 2.3, the URL/filename/text might be prefixed with `GeoJSON:` to avoid any ambiguity with other drivers.

5.28.3 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.

Starting with GDAL 2.2, 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 new layer. This behavior conforms to 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 consists 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/vnd.geo+json” in the `NATIVE_MEDIA_TYPE` of the `NATIVE_DATA` metadata domain).

5.28.4 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. Starting with GDAL 2.0, Integer(Boolean), Date, Time and DateTime fields are also recognized.

It is possible to tell the driver to not to process attributes by setting environment variable **ATTRIBUTES_SKIP=YES**. Default behavior is to preserve all attributes (as an union, see previous paragraph), what 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/vnd.geo+json” in the NativeMediaType property). On write, if a OGRFeature to be written has its NativeMediaType property set to “application/vnd.geo+json” 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.

5.28.5 Geometry

Similarly to the issue with mixed-properties features, the *GeoJSON Specification* draft does not require all *Feature* objects in a collection must have geometry of the same type. 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 there is a need to generate 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 **GEOMETRY_AS_COLLECTION=YES** in the environment (default is **NO**).

5.28.6 Environment variables

- **GEOMETRY_AS_COLLECTION** - used to control translation of geometries: YES - wrap geometries with OGRGeometryCollection type
- **ATTRIBUTES_SKIP** - controls translation of attributes: YES - skip all attributes
- **OGR_GEOJSON_MAX_OBJ_SIZE** - (GDAL >= 3.0.2) size in MBytes of the maximum accepted single feature, default value is 200MB

5.28.7 Open options

(GDAL >= 2.0)

- **FLATTEN_NESTED_ATTRIBUTES** = YES/NO: Whether to recursively explore nested objects and produce flatten OGR attributes. Defaults to NO.
- **NESTED_ATTRIBUTE_SEPARATOR** = character: Separator between components of nested attributes. Defaults to '_'
- **FEATURE_SERVER_PAGING** = YES/NO: Whether to automatically scroll through results with a ArcGIS Feature Service endpoint.
- **NATIVE_DATA** = YES/NO: (GDAL >= 2.1) Whether to store the native JSon representation at FeatureCollection and Feature level. Defaults to NO. 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: (GDAL >= 2.1) 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) 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.

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",
        "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)
```

5.28.8 Layer creation options

- **WRITE_BBOX** = YES/NO: (OGR >= 1.9.0) Set to YES to write a bbox property with the bounding box of the geometries at the feature and feature collection level. Defaults to NO.
- **COORDINATE_PRECISION** = int_number: (OGR >= 1.9.0) 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** = int_number: (OGR >= 2.1) Maximum number of significant figures when writing floating-point numbers. Default to 17. If explicitly specified, and **COORDINATE_PRECISION** is not, this will also apply to coordinates.
- **NATIVE_DATA**=string. (OGR >= 2.1) Serialized JSon object that contains extra properties to store at FeatureCollection level.
- **NATIVE_MEDIA_TYPE**=string. (OGR >= 2.1) Format of **NATIVE_DATA**. Must be “application/vnd.geo+json”, otherwise **NATIVE_DATA** will be ignored.
- **RFC7946**=YES/NO. (OGR >= 2.2) Whether to use [RFC 7946](#) standard. Otherwise [GeoJSON 2008](#) initial version will be used. Default is NO (thus GeoJSON 2008)
- **WRITE_NAME**=YES/NO. (OGR >= 2.2) Whether to write a “name” property at feature collection level with layer name. Defaults to YES.
- **DESCRIPTION**=string. (OGR >= 2.2) (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**=string. (OGR >= 2.3) Name of the source field that must be written as the ‘id’ member of Feature objects.
- **ID_TYPE**=AUTO/String/Integer. (OGR >= 2.3) Type of the ‘id’ member of Feature objects.
- **ID_GENERATE**=YES/NO. (OGR >= 3.1) Auto-generate feature ids
- **WRITE_NON_FINITE_VALUES**=YES/NO. (OGR >= 2.4) 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. Defaults to NO

5.28.9 VSI Virtual File System API support

Some features below require OGR >= 1.9.0.

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), `/vsizip/` (read-write), `/vsicurl/` (read-only) domains.

Writing to `/dev/stdout` or `/vsistdout/` is also supported.

5.28.10 Round-tripping of extra JSon members

See [rfc-60](#) for more details.

Starting with GDAL 2.1, 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 behaviour can be turned off by specifying the **-noNativeData** switch of the ogr2ogr utility.

5.28.11 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.

5.28.12 Examples

How to dump content of .geojson file:

```
ogrinfo -ro point.geojson
```

How to query features from remote service with filtering by attribute:

```
ogrinfo -ro http://featureserver/cities/.geojson OGRGeoJSON -where "name=Warsaw"
```

How to translate number of features queried from FeatureServer to ESRI Shapefile:

```
ogr2ogr -f "ESRI Shapefile" cities.shp http://featureserver/cities/.geojson OGRGeoJSON
```

How to translate a ESRI Shapefile into a RFC 7946 GeoJSON file:

```
ogr2ogr -f GeoJSON cities.json cities.shp -lco RFC7946=YES
```

5.28.13 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*
- *ESRI JSON / FeatureService driver*
- *TopoJSON driver*

5.29 GeoJSONSeq: sequence of GeoJSON features

New in version 2.4.

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.

5.29.1 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.*)

5.29.2 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.

5.29.3 Layer creation options

- **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** = int_number: Maximum number of figures after decimal separator to write in coordinates. Default to 7. “Smart” truncation will occur to remove trailing zeros.
- **SIGNIFICANT_FIGURES** = int_number: Maximum number of significant figures when writing floating-point numbers. Default to 17. If explicitly specified, and `COORDINATE_PRECISION` is not, this will also apply to coordinates.
- **ID_FIELD**=string. 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.

5.29.4 See Also

- [GeoJSON driver](#)
- [RFC 7946](#) standard: the GeoJSON Format.
- [RFC 8142](#) standard: GeoJSON Text Sequences (RS separator)
- [Newline Delimited JSON](#)
- [GeoJSONL](#): An optimized format for large geographic datasets

5.30 Geomedia MDB database

Driver short name

Geomedia

Build dependencies

ODBC library

OGR optionally supports reading Geomedia .mdb files via ODBC. Geomedia is a Microsoft Access database with a set of tables defined by Intergraph for holding geodatabase metadata, and with geometry for features held in a BLOB column in a custom format. This driver accesses the database via ODBC but does not depend on any Intergraph middle-ware.

Geomedia .mdb are accessed by passing the file name of the .mdb file to be accessed as the data source name. On Windows, no ODBC DSN is required. On Linux, there are problems with DSN-less connection due to incomplete or buggy implementation of this feature in the [MDB Tools](#) package. So, it is required to configure Data Source Name (DSN) if the MDB Tools driver is used (check instructions below).

In order to facilitate compatibility with different configurations, the GEOMEDIA_DRIVER_TEMPLATE Config Option was added to provide a way to programmatically set the DSN programmatically 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 (arcs are not yet). Coordinate system information is not currently supported.

Currently the OGR Personal Geodatabase driver does not take advantage of spatial indexes for fast spatial queries.

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.

5.30.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

5.30.2 How to use Geomedia driver with unixODBC and MDB Tools (on Unix and Linux)

Starting with GDAL/OGR 1.9.0, the [MDB](#) driver is an alternate way of reading Geomedia .mdb files without requiring unixODBC and MDB Tools

Refer to the similar section of the [PGeo](#) driver. The prefix to use for this driver is Geomedia:

5.30.3 See also

- [MDB](#) driver page

5.31 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.

5.31.1 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.*)

5.31.2 Encoding issues

Expat library supports reading the following built-in encodings :

- US-ASCII
- UTF-8
- UTF-16
- ISO-8859-1

OGR 1.8.0 adds supports for Windows-1252 encoding (for previous versions, altering the encoding mentioned in the XML header to ISO-8859-1 might work in some cases).

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.

5.31.3 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)
```

5.31.4 Creation Issues

On export, all layers are written to a single file. Update of existing files is not supported.

If the output file already exists, 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.

The GeorSS writer supports the following *dataset* creation options:

- **FORMAT=RSS|ATOM**: whether the document must be in RSS 2.0 or Atom 1.0 format. Default value : RSS
- **GEOM_DIALECT=SIMPLE|GML|W3C_GEO** (RSS or ATOM document): the encoding of location information. Default value : SIMPLE W3C_GEO only supports point geometries. SIMPLE or W3C_GEO only support geometries in geographic WGS84 coordinates.
- **USE_EXTENSIONS=YES|NO**. Default value : 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**. Default value : 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** (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** (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** (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** (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** (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** (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** (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](#))

5.31.5 VSI Virtual File System API support

(Some features below might require OGR >= 1.9.0)

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.

5.31.6 Example

The ogrinfo utility can be used to dump the content of a GeoRSS datafile :

```
ogrinfo -ro -al input.xml
```

The ogr2ogr utility can be used to do GeoRSS to GeoRSS translation. For example, to translate a Atom document into a RSS document

```
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"
```

Note : in this example we map equivalent fields, from the source name to the expected name of the destination format.

The following Python script shows how to read the content of a online GeoRSS feed

```
#!/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:
    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)
```

(continues on next page)

(continued from previous page)

```
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')
```

5.31.7 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](#)

5.32 GMLAS - Geography Markup Language (GML) driven by application schemas

New in version 2.2.

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.

5.32.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

5.32.2 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.

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.

5.32.3 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 convenience 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.

5.32.4 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.

5.32.5 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. Disabled 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 XPath 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).

5.32.6 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.

5.32.7 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.

5.32.8 Open options

- **XSD=filename(s)**: 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** or inline XML definition: 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**: whether the metadata layers “_ogr_fields_metadata”, “_ogr_layers_metadata”, “_ogr_layer_relationships” and “ogr_other_metadata” should be reported by default. Default is NO.
- **VALIDATE=YES/NO**: whether the document should be validated against the schemas. Validation is done at dataset opening. Default is NO.
- **FAIL_IF_VALIDATION_ERROR=YES/NO**: Whether a validation error should cause dataset opening to fail. (only used if **VALIDATE=YES**) Default is NO.
- **REFRESH_CACHE=YES/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. Default is NO.
- **SWAP_COORDINATES=AUTO/YES/NO**: 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. The default is AUTO.
- **REMOVE_UNUSED_LAYERS=YES/NO**: Whether unused layers should be removed from the reported layers. Defaults to NO.

- **REMOVE_UNUSED_FIELDS=YES/NO**: Whether unused fields should be removed from the reported layers. Defaults to NO
- **HANDLE_MULTIPLE_IMPORTS=YES/NO**: Whether multiple imports with the same namespace but different schema are allowed. Defaults to NO
- **SCHEMA_FULL_CHECKING=YES/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. Defaults to NO

5.32.9 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-driven, 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.

5.32.9.1 ogr2ogr behaviour

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.

5.32.9.2 Dataset creation options

The supported dataset creation options are:

- **INPUT_XSD**=filename(s): 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 or inline XML definition: 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**=layers. 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]. Number of spaces for each indentation level. Default is 2.
- **COMMENT**=string. 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. Whether to wrap features in a `wfs:FeatureCollection` or in a `ogr_gmlas:FeatureCollection`. Defaults to WFS2_FEATURECOLLECTION.
- **TIMESTAMP**=XML date time. 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**=Path or URL to `wfs.xsd`. Only valid for WRAPPING=WFS2_FEATURECOLLECTION. Default is “[http://schemas.opengis.net/wfs/2.0/wfs.xsd](#)”
- **GENERATE_XSD**=YES/NO. 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. Default to YES.
- **OUTPUT_XSD_FILENAME**=string. 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

5.32.10 Examples

Listing content of a data file:

```
ogrinfo -ro GMLAS:my.gml
```

Converting to PostGIS:

```
ogr2ogr -f PostgreSQL PG:'host=myserver dbname=warmerda' GMLAS:my.gml -nlt CONVERT_TO_↵LINEAR
```

Converting to Spatialite and back to GML

```
ogr2ogr -f SQLite tmp.sqlite GMLAS:in.gml -dsco SPATILIATE=YES -nlt CONVERT_TO_LINEAR_↵↵-oo EXPOSE_METADATA_LAYERS=YES
ogr2ogr -f GMLAS out.gml tmp.sqlite
```

5.32.11 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

5.32.12 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).

5.32.12.1 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
<pre> <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> </pre>	<pre> <MyFeature id="my_ ↪ id" attr="attr_ ↪ value"> <name>my_name</ ↪ name> </MyFeature> </pre>	<pre> Layer name:_ ↪ MyFeature Geometry: None id: String (0.0)_ ↪ NOT NULL attr: String (0.0) name: String (0.0)_ ↪ NOT NULL OGRFeature(MyFeature):1 id (String) =_ ↪ my_id attr (String)_ ↪ = attr_value name (String)_ ↪ = my_name </pre>	Element with attributes and sub-elements of simple type and maximum cardinality of 1.
<pre> <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> </pre>	<pre> <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> </pre>	<pre> Layer name:_ ↪ MyFeature Geometry: None id: String (0.0)_ ↪ NOT NULL str_array:_ ↪ StringList (0.0)_ ↪ NOT NULL OGRFeature(MyFeature):1 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 value_ ↪ (DateTime) =_ ↪ 2016/09/24_ </pre>	Example with array and child layer
594		<pre> parent_id_ ↪ (String) = my_id value_ ↪ (DateTime) =_ ↪ 2016/09/24_ </pre>	Chapter 5. Vector drivers

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
Layer SRS WKT:
(unknown)
parent_ogr_pkid: String (0.0) NOT NULL
```

(continues on next page)

(continued from previous page)

```
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](#)

5.32.12.2 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 structure is:

Field name	Description
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_behavior	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 structure is:

Field name	Description
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 structure is:

Field name	Description
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 structure is:

Field name	Description
key	Name of the metadata item
value	Value of the metadata item

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](#)*

5.33 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)

Starting with GDAL 2.2, another driver, [GMLAS](#), for GML driven by application schemas, is also available. Both GML and GMLAS drivers have their use cases.

5.33.1 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.*)

5.33.2 Parsers

The reading part of the driver only works if OGR is built with Xerces linked in. Starting with OGR 1.7.0, 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: starting with OGR 1.9.0, 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**.

5.33.3 CRS support

Since OGR 1.8.0, 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:](http://www.opengis.net/def/crs/EPSSG/0)” (or “<http://www.opengis.net/def/crs/EPSSG/0>” starting with GDAL 2.1.2), 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.

Since OGR 1.10, the above also applied for projected coordinate systems whose EPSG preferred axis order is (northing, easting).

Starting with GDAL 2.1.2, 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`.

5.33.4 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.

Starting with OGR 1.10, 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. Starting with GDAL 2.0, 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 be parsed correctly, it is completely scanned in order to determine the set of featurtypes, 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**.

Starting with GDAL 1.11, 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.

Configuration options can be set via the CPLSetConfigOption() function or as environment variables.

5.33.5 Particular GML application schemas

OGR 1.8.0 adds support for detecting feature attributes in nested GML elements (non-flat attribute hierarchy) that can be found in some GML profiles such as UK Ordnance Survey MasterMap. OGR 1.8.0 also brings support for reading IntegerList, RealList and StringList field types when a GML element has several occurrences.

Since OGR 1.8.0, a specialized GML driver - the *NAS* driver - is available to read German AAA GML Exchange Format (NAS/ALKIS).

Since OGR 1.8.0, the GML driver has partial support for reading AIXM or CityGML files.

Since OGR 1.11, 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).

Since OGR 2.0, the GML driver supports reading responses to CSW GetRecords queries.

Since OGR 2.2, the GML driver supports reading Japanese FGD GML v4 files.

5.33.6 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, starting with OGR 1.11, 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.

Starting with OGR 1.10, in case of multiple geometry occurrences, if a geometry is in a <geometry> element, this will be the one selected. This will make default behaviour consistent with Inspire objects.

Starting with OGR 1.8.0, 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.

OGR 1.8.0 adds support for more GML geometries including TopoCurve, TopoSurface, MultiCurve. 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.

Starting with GDAL 2.0, 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.

5.33.7 gml:xlink resolving

OGR 1.8.0 adds support for gml:xlink resolving. 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 behaviour 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.

Starting since OGR 1.9.0 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:
 - you can use **GML_GFS_TEMPLATE 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.

5.33.8 TopoSurface interpretation rules [polygons and internal holes]

Starting since OGR 1.9.0 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 anyway 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

5.33.9 Encoding issues

Expat library supports reading the following built-in encodings :

- US-ASCII
- UTF-8
- UTF-16
- ISO-8859-1

When used with Expat library, OGR 1.8.0 adds supports for Windows-1252 encoding (for previous versions, altering the encoding mentioned in the XML header to ISO-8859-1 might work in some cases).

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.

5.33.10 Feature id (fid / gml:id)

Starting with OGR 1.8.0, 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.

Starting with OGR 1.9.0, 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**.

Starting with OGR 1.9.0, 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.

5.33.11 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.

Starting with OGR 1.9.0, 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.

Starting with OGR 1.9.0, 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);
```

5.33.12 Open options

- **XSD=filename:** (GDAL >=2.0) to specify an explicit filename for the XSD application schema to use.
- **FORCE_SRS_DETECTION=YES/NO:** (GDAL >=2.0) 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. Defaults to NO
- **EMPTY_AS_NULL=YES/NO:** (GDAL >=2.0) 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 behaviour. 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:** (GDAL >=2.0) 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. Defaults to NO.
- **INVERT_AXIS_ORDER_IF_LAT_LONG=YES/NO:** (GDAL >=2.0) Whether to present SRS and coordinate ordering in traditional GIS order. Defaults to YES.
- **CONSIDER_EPSG_AS_URN=YES/NO/AUTO:** (GDAL >=2.0) Whether to consider srsName like `EPSG:XXXX` as respecting EPSG axis order. Defaults to AUTO.
- **SWAP_COORDINATES=AUTO/YES/NO:** (GDAL >=2.1.2) 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`.
- **READ_MODE=AUTO/STANDARD/SEQUENTIAL_LAYERS/INTERLEAVED_LAYERS:** (GDAL >=2.0) Read mode. Defaults to `AUTO`.

- **EXPOSE_GML_ID=YES/NO/AUTO:** (GDAL >=2.0) Whether to make feature gml:id as a gml_id attribute. Defaults to AUTO.
- **EXPOSE_FID=YES/NO/AUTO:** (GDAL >=2.0) Whether to make feature fid as a fid attribute. Defaults to AUTO.
- **DOWNLOAD_SCHEMA=YES/NO:** (GDAL >=2.0) Whether to download the remote application schema if needed (only for WFS currently). Defaults to YES.
- **REGISTRY=filename:** (GDAL >=2.0) Filename of the registry with application schemas. Defaults to {GDAL_DATA}/gml_registry.xml.

5.33.13 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.

The GML writer supports the following dataset creation options:

- **XSISHEMAURI:** 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.
- **XSISHEMA:** This can be EXTERNAL, INTERNAL or OFF and defaults to 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 XSISHEMAURI is used).
- **PREFIX** (OGR >= 1.10) Defaults to 'ogr'. This is the prefix for the application target namespace.
- **STRIP_PREFIX** (OGR >= 1.11) Defaults to FALSE. Can be set to TRUE to avoid writing the prefix of the application target namespace in the GML file.
- **TARGET_NAMESPACE** (OGR >= 1.10) Defaults to '<http://ogr.maptools.org/>'. This is the application target namespace.
- **FORMAT:** (OGR >= 1.8.0) This can be set to :
 - *GML3* in order to write GML files that follow GML 3.1.1 SF-0 profile.
 - *GML3Deegree* (OGR >= 1.9.0) 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*(OGR >= 1.9.0) in order to write GML files that follow GML 3.2.1 SF-0 profile.

If not specified, GML2 will be used. Starting with GDAL 2.0, 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: starting with OGR 1.11, fields of type StringList, RealList or IntegerList can be written. This will cause to advertize the SF-1 profile in the .XSD schema (such types are not supported by SF-0).

- **GML_FEATURE_COLLECTION=YES/NO** (OGR >= 2.3) 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. (OGR >= 1.8.0, 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 (Only valid for FORMAT=GML3/GML3Degree/GML3.2, GDAL >= 2.2). Defaults to OGC_URN. 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, GDAL >= 2.0) 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. (OGR >= 1.11, only valid when FORMAT=GML3/GML3Degree/GML3.2) Default to YES. If set to NO, the <gml:boundedBy> element will not be written for each feature.
- **SPACE_INDENTATION**=YES/NO. (OGR >= 1.8.0) Default to YES. If YES, the output will be indented with spaces for more readability, but at the expense of file size.
- **GML_ID**=string. (Only valid for GML 3.2, GDAL >= 2.0) Value of feature collection gml:id. Default value is “aFeatureCollection”.
- **NAME**=string. Content of GML name element. Can also be set as the NAME metadata item on the dataset.
- **DESCRIPTION**=string. Content of GML description element. Can also be set as the DESCRIPTION metadata item on the dataset.

5.33.14 VSI Virtual File System API support

(Some features below might require OGR >= 1.9.0)

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/

5.33.15 Syntax of .gfs file by example

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>
      <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 only supported from OGR 1.8.0, 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).

Starting with OGR 1.11, 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)

```

5.33.16 Advanced .gfs syntax (OGR >= 1.11)

5.33.16.1 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>
        </Object>
      </content>
    </TopLevelObject>
  </gml:featureMember>
</gml:FeatureCollection>

```

(continues on next page)

(continued from previous page)

```

        </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 behaviour 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>
    <PropertyDefn>
      <Name>foo</Name>
      <ElementPath>foo</ElementPath>
      <Type>String</Type>
    </PropertyDefn>
  </GMLFeatureClass>
</GMLFeatureClassList>

```

5.33.16.2 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>

```

(continues on next page)

(continued from previous page)

```

    </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)
length_unit: String (0.0)
OGRFeature(LAYER):0
  gml_id (String) = (null)
  length (Real) = 5
  length_unit (String) = m

```

5.33.16.3 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>

```

(continues on next page)

(continued from previous page)

```

    <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>
      <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

```

(continues on next page)

(continued from previous page)

```

name_fr (String) = Nom francais
name_others_lang (StringList) = (1:de)
name_others (StringList) = (1:Deutsche name)

```

5.33.17 Registry for GML application schemas (OGR >= 1.11)

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"/>
    <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">

```

(continues on next page)

(continued from previous page)

```

    <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:ktjkiwfs="http://xml.nls.fi/ktjkiwfs/2010/02"*), and that the feature type is also detected in the first bytes of the GML file (e.g. *ktjkiwfs:KiinteistorajanSijaintitiedot*). If the element value is defined then 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*).

5.33.18 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
  [...]
  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

```
python 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

5.33.19 Reading datasets resulting from a WFS 2.0 join queries

Starting with GDAL 2.0, 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></
↪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
```

(continues on next page)

(continued from previous page)

```

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)

```

5.33.20 Examples

The ogr2ogr utility can be used to dump the results of a Oracle query to GML:

```
ogr2ogr -f GML output.gml OCI:usr/pwd@db my_feature -where "id = 0"
```

The ogr2ogr utility can be used to dump 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'"
```

5.33.21 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](#)

5.33.22 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)

5.34 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.

5.34.1 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.)*

5.34.2 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.

5.35 GPKG – GeoPackage vector

Driver short nameGPKG

Build dependencieslibsqlite3

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.

Starting with GDAL 2.0, the driver also supports reading and writing the following non-linear geometry types: CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE and MULTISURFACE

Starting with GDAL 2.0, GeoPackage raster/tiles are supported. See [GeoPackage raster](#) documentation page

5.35.1 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.*)

5.35.2 Specification version

Starting with GDAL 2.2, the driver is able to create GeoPackage databases following the 1.0/1.0.1, 1.1 or 1.2 versions. For GDAL 2.2, it will automatically adjust to the minimum version required for the features of GeoPackage used. For GDAL 2.3 or later, it will default to 1.2. Explicit version choice can be done by specifying the VERSION dataset creation option.

5.35.3 Limitations

- GeoPackage only supports one geometry column per table.

5.35.4 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.

Starting with GDAL 2.0, SQL SELECT statements passed to ExecuteSQL() are also executed directly against the database. If Spatialite is used, a recent version (4.2.0) is needed and use of explicit cast operators AsGPB() is required to transform GeoPackage geometries to Spatialite geometries (the reverse conversion from Spatialite geometries is automatically done by the GPKG driver). It is also possible to use with any Spatialite version, but in a slower way, by specifying the “INDIRECT_SQLITE” dialect. In which case, GeoPackage geometries automatically appear as Spatialite geometries after translation by OGR.

Starting with GDAL 2.2, the “DROP TABLE layer_name” and “ALTER TABLE layer_name RENAME TO new_layer” statements can be used. They will update GeoPackage system tables.

Starting with GDAL 2.2, 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.

5.35.4.1 SQL functions

Starting with GDAL 2.0, 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

5.35.4.2 Link with Spatialite

Starting with GDAL 2.0, if it has been compiled against Spatialite 4.2 or later, it is also possible to use Spatialite SQL functions. Explicit transformation from GPKG geometry binary encoding to Spatialite geometry binary encoding must be done.

```
ogrinfo poly.gpkg -sql "SELECT ST_Buffer(CastAutomagic(geom),5) FROM poly"
```

Starting with Spatialite 4.3, `CastAutomagic` is no longer needed.

5.35.5 Transaction support (GDAL >= 2.0)

The driver implements transactions at the database level, per rfc-54

5.35.6 Opening options

The following open options are available:

- **LIST_ALL_TABLES=**AUTO/YES/NO: (GDAL >=2.2) Whether all tables, including those not listed in `gpkg_contents`, should be listed. Defaults to AUTO. 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.

Note: open options are typically specified with “-oo name=value” syntax in most OGR utilities, or with the `GDALOpenEx()` API call.

5.35.7 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.

5.35.7.1 Dataset Creation Options

The following creation options (specific to vector, or common with raster) are available:

- **VERSION=**AUTO/1.0/1.1/1.2: (GDAL >= 2.2) Set GeoPackage version (for `application_id` and `user_version` fields). In AUTO mode, this will be equivalent to 1.2 starting with GDAL 2.3.
- **ADD_GPKG_OGR_CONTENTS=**YES/NO: (GDAL >= 2.2) Defines whether to add a `gpkg_ogr_contents` table to keep feature count, and associated triggers. Defaults to YES.

Other options are available for raster. See the [GeoPackage raster](#) documentation page

5.35.7.2 Layer Creation Options

- **GEOMETRY_NAME**: Column to use for the geometry column. Default to “geom”. Note: This option was called **GEOMETRY_COLUMN** in releases before GDAL 2
- **GEOMETRY_NULLABLE**: (GDAL >=2.0) Whether the values of the geometry column can be NULL. Can be set to NO so that geometry is required. Default to “YES”
- **FID**: Column name to use for the OGR FID (primary key in the SQLite database). Default to “fid”
- **OVERWRITE**: If set to “YES” will delete any existing layers that have the same name as the layer being created. Default to NO
- **SPATIAL_INDEX**: (GDAL >=2.0) If set to “YES” will create a spatial index for this layer. Default to YES
- **PRECISION**: (GDAL >=2.0) 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. The default is “YES”.
- **TRUNCATE_FIELDS**: (GDAL >=2.0) 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. The default is “NO”.
- **IDENTIFIER=string**: (GDAL >=2.0) Identifier of the layer, as put in the contents table.
- **DESCRIPTION=string**: (GDAL >=2.0) Description of the layer, as put in the contents table.
- **ASPATIAL_VARIANT=GPKG_ATTRIBUTES/OGR_ASPATIAL/NOT_REGISTERED**: (GDAL >=2.2) How to register non spatial tables. Defaults to GPKG_ATTRIBUTES in GDAL 2.2 or later (behaviour in previous version was equivalent to OGR_ASPATIAL). 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. This is only compatible of GDAL 2.2 or later. Priorly, in OGR 2.0 and 2.1, the “aspatial” extension had been developed for similar purposes, so if selecting OGR_ASPATIAL, non spatial tables will be recorded with data_type=”aspatial” and the “aspatial” extension was declared in the gpkg_extensions 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.

5.35.8 Metadata

(GDAL >=2.0) 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.

5.35.8.1 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.

Starting with GDAL 2.0, the driver allows creating and reading non-spatial tables with the *GeoPackage aspatial extension*.

Starting with GDAL 2.2, 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. Starting with GDAL 2.2, non spatial tables are by default created following the GeoPackage 1.2 “attributes” data type (can be controlled with the `ASPIATIAL_VARIANT` layer creation option)

5.35.9 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.

Starting with GDAL 2.2, 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. Starting with GDAL 2.3, this can be easily verified if the `SQLITE_HAS_COLUMN_METADATA=YES` driver metadata item is declared (for example with “`ogrinfo -format GPKG`”)

5.35.10 Level of support of GeoPackage Extensions

(Restricted to those have a vector scope)

Table 3: Extensions

Extension name	OGC adopted extension ?	Supported by GDAL?
Non-Linear Geometry Types	Yes	Yes, since GDAL 2.1
RTree Spatial Indexes	Yes	Yes, since GDAL 2.0
Metadata	Yes	Yes, since GDAL 1.11
Schema	Yes	No
WKT for Coordinate Reference Systems (WKT v2)	Yes	Partially, since GDAL 2.2. GDAL can read databases using this extension, but cannot interpret a SRS entry that has only a WKT v2 entry.
<i>GeoPackage aspatial extension</i>	No	Yes, since GDAL 2.0. Deprecated in GDAL 2.2 for the <i>attributes</i> official data_type

5.35.11 Examples

- 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
```

- Translation of 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
```

- Translation of 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'
```

5.35.12 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](#)

5.35.12.1 GeoPackage aspatial extension

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.

This was used in GDAL 2.0 and GDAL 2.1, before the introduction of the ‘attributes’ `data_type` of GeoPackage v1.2. Starting with GDAL 2.2, ‘attributes’ will be used by default, so this extension is now legacy.

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
<i>id</i>	INTEGER	Autoincrement primary key	no		PK
<i>text_attribute</i>	TEXT	Text attribute of row	yes		
<i>real_attribute</i>	REAL	Real attribute of row	yes		
<i>boolean_attribute</i>	BOOLEAN	Boolean attribute of row	yes		
<i>raster_or_photo</i>	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

5.36 GPSTable

Driver short name

GPSTable

Build dependencies

(read support needs GPX driver and libexpat)

The GPSTable driver for now that relies on the [GPSTable](#) utility to access various GPS file formats.

The GPSTable executable must be accessible through the PATH.

5.36.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

5.36.2 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 GPSTable, 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 : `GPSTable:gpsbabel_file_format[,gpsbabel_format_option]:[features=[waypoints,][tracks,]` where :

- `gpsbabel_file_format` is one of the [file formats](#) handled by GPSTable.
- `gpsbabel_format_option` is any option handled by the specified GPSTable format (refer to the documentation of each GPSTable format)
- `features=` can be used to modify the type of features that GPSTable 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 GPSTable 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 GPSTBabel such as “usb:”, “/dev/ttyS0”, “COM1:”, etc.. What is actually supported depends on the used GPSTBabel format.

Alternatively, for a few selected GPSTBabel formats, just specifying the filename might be sufficient. The list includes for now :

- garmin_txt
- 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.

5.36.3 Write support

The driver relies on the GPX driver to create an intermediate file that will be finally translated by GPSTBabel to the desired GPSTBabel format. (The GPX driver does not need to be configured for read support for GPSTBabel 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 : *GPSTBabel:gpsbabel_file_format[,gpsbabel_format_option]:filename** where :

- *gpsbabel_file_format* is one of the [file formats](#) handled by GPSTBabel.
- *gpsbabel_format_option* is any option handled by the specified GPSTBabel format (refer to the documentation of each GPSTBabel format)

Alternatively, you can just pass a filename as output datasource name and specify the dataset creation option GPSTBABEL_DRIVER=gpsbabel_file_format[,gpsbabel_format_option]*

The USE_TEMPFILE=YES configuration option can be used to create an on-disk temporary GPX file instead of a in-memory one, when writing big amount of data.

5.36.3.1 Examples

Reading the waypoints from a Garmin USB receiver :

```
ogrinfo -ro -al GPSTables:garmin:usb:
```

Converting a shapefile to Magellan Mapsend format :

```
ogr2ogr -f GPSTables GPSTables:mapsend:out.mapsend in.shp
```

5.36.3.2 See Also

- [GPSTables Home Page](#)
- [GPSTables file formats](#)
- [GPX driver page](#)

5.37 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).

If the environment variable `GPX_ELE_AS_25D` is set to YES, the elevation element will be used to set the Z coordinates of waypoints, route points and track points.

The OGR/GPX reads and writes the GPX attributes for the waypoints, routes and tracks.

By default, up to 2 <link> elements can be taken into account by feature. This default number can be changed with the `GPX_N_MAX_LINKS` environment variable.

5.37.1 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.*)

5.37.2 Encoding issues

Expat library supports reading the following built-in encodings :

- US-ASCII
- UTF-8
- UTF-16
- ISO-8859-1

OGR 1.8.0 adds supports for Windows-1252 encoding (for previous versions, altering the encoding mentioned in the XML header to ISO-8859-1 might work in some cases).

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.

5.37.3 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** environment variable.

5.37.4 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 exists, 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.

Starting with GDAL/OGR 1.8.0, 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.

The GPX writer supports the following *layer* creation options:

- **FORCE_GPX_TRACK:** By default when writing a layer whose features are of type wkbLineString, the GPX driver chooses to write them as routes. If FORCE_GPX_TRACK=YES is specified, they will be written as tracks.
- **FORCE_GPX_ROUTE:** By default when writing a layer whose features are of type wkbMultiLineString, the GPX driver chooses to write them as tracks. If FORCE_GPX_ROUTE=YES is specified, they will be written as routes, provided that the multilines are composed of only one single line.

The GPX writer supports the following *dataset* creation options:

- **GPX_USE_EXTENSIONS:** 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:** Only used if GPX_USE_EXTENSIONS=YES and GPX_EXTENSIONS_NS_URL is set. The namespace value used for extension tags. By default, “ogr”.
- **GPX_EXTENSIONS_NS_URL:** Only used if GPX_USE_EXTENSIONS=YES and GPX_EXTENSIONS_NS is set. The namespace URI. By default, “<http://osgeo.org/gdal>”.
- **LINEFORMAT:** (GDAL/OGR >= 1.8.0) 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 layer creation option which may have a value of **CRLF** (DOS format) or **LF** (Unix format).

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.

5.37.5 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, starting with GDAL 1.6.1, you can define the GPX_SHORT_NAMES configuration option to TRUE to make them be reported respectively as “trksegid” and “trksegptid”, which will allow them to be unique once translated to DBF. The “route_point_id” field of *route_points* layer will also be renamed to “rteptid”. But note that no particular processing will be done for any extension field names.

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.

Starting with GDAL 1.6.0, 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
↪"
```

Starting with GDAL 1.7.0, 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
```

5.37.6 VSI Virtual File System API support

(Some features below might require OGR >= 1.9.0)

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.

5.37.7 Example

The ogrinfo utility can be used to dump the content of a GPX datafile :

```
ogrinfo -ro -al input.gpx
```

The ogr2ogr utility can be used to do 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.

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">
```

(continues on next page)

(continued from previous page)

```

<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>

```

Use of -sql option to remap field names to the ones allowed by the GPX schema (starting with GDAL 1.6.0):

```
ogr2ogr -f GPX output.gpx input.shp -sql "SELECT field1 AS name, field2 AS desc FROM_
↪input"
```

5.37.8 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).

5.37.9 See Also

- [Home page for GPX format](#)
- [GPX 1.1 format documentation](#)

5.38 GRASS Vector Format

Driver short name

GRASS

Build dependencies

libgrass

GRASS driver can read GRASS (version 6.0 and higher) vector maps. Each GRASS vector map is represented as one datasource. A GRASS vector map may have 0, 1 or more layers.

GRASS points are represented as wkbPoint, lines and boundaries as wkbLineString and areas as wkbPolygon. wkb-Multi* and wkbGeometryCollection are not used. More feature types can be mixed in one layer. If a layer contains only features of one type, it is set appropriately and can be retrieved by OGRLayer::GetLayerDefn();

If a geometry has more categories of the same layer attached, its represented as more features (one for each category). Both 2D and 3D maps are supported.

5.38.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

5.38.2 Datasource name

Datasource name is full path to 'head' file in GRASS vector directory. Using names of GRASS environment variables it can be expressed:

```
$GISDBASE/$LOCATION_NAME/$MAPSET/vector/mymap/head
```

where 'mymap' is name of a vector map. For example:

```
/home/cimrman/grass_data/jizerky/jara/vector/liptakov/head
```

5.38.3 Layer names

Usually layer numbers are used as layer names. Layer number 0 is used for all features without any category. It is possible to optionally give names to GRASS layers linked to database however currently this is not supported by grass modules. A layer name can be added in 'dbln' vector file as '/name' after layer number, for example to original record:

```
1 rivers cat $GISDBASE/$LOCATION_NAME/$MAPSET/dbf/ dbf
```

it is possible to assign name 'rivers'

```
1/rivers rivers cat $GISDBASE/$LOCATION_NAME/$MAPSET/dbf/ dbf
```

the layer 1 will be listed is layer 'rivers'.

5.38.4 Attribute filter

If a layer has attributes stored in a database, the query is passed to the underlying database driver. That means, that SQL conditions which can be used depend on the driver and database to which the layer is linked. For example, DBF driver has currently very limited set of SQL expressions and PostgreSQL offers very rich set of SQL expressions.

If a layer has no attributes linked and it has only categories, OGR internal SQL engine is used to evaluate the expression. Category is an integer number attached to geometry, it is sort of ID, but it is not FID as more features in one layer can have the same category.

Evaluation is done once when the attribute filter is set.

5.38.5 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.

5.38.6 GISBASE

GISBASE is full path to the directory where GRASS is installed. By default, GRASS driver is using the path given to gdal configure script. A different directory can be forced by setting GISBASE environment variable. GISBASE is used to find GRASS database drivers.

5.38.7 Missing topology

GRASS driver can read GRASS vector files if topology is available (AKA level 2). If an error is reported, telling that the topology is not available, it is necessary to build topology within GRASS using v.build module.

5.38.8 Random access

If random access (GetFeature instead of GetNextFeature) is used on layer with attributes, the reading of features can be quite slow. It is because the driver has to query attributes by category for each feature (to avoid using a lot of memory) and random access to database is usually slow. This can be improved on GRASS side optimizing/writing file based (DBF, SQLite) drivers.

5.38.9 Known problem

Because of bug in GRASS library, it is impossible to start/stop database drivers in FIFO order and FILO order must be used. The GRASS driver for OGR is written with this limit in mind and drivers are always closed if not used and if a driver remains opened kill() is used to terminate it. It can happen however in rare cases, that the driver will try to stop database driver which is not the last opened and an application hangs. This can happen if sequential read (GetNextFeature) of a layer is not finished (reading is stopped before last available feature is reached), features from another layer are read and then the reading of the first layer is finished, because in that case kill() is not used.

5.38.10 See Also

- [GRASS GIS home page](#)

Development of this driver was financially supported by Faunalia (www.faunalia.it).

5.39 GTM - GPS TrackMaker

Driver short name

GTM

Driver built-in by default

This driver is built-in by default

GPSTrackMaker is a program that is compatible with more than 160 GPS models. It allows you to create your own maps. It supports vector maps and images.

The OGR driver has support for reading and writing GTM 211 files (.gtm); however, in this implementation we are not supporting images and routes. Waypoints and tracks are supported.

Although GTM has support for many data, like NAD 1967, SAD 1969, and others, the output file of the OGR driver will be using WGS 1984. And the GTM driver will only read properly GTM files georeferenced as WGS 1984 (if not the case a warning will be issued).

The OGR driver supports just POINT, LINESTRING, and MULTILINESTRING.

5.39.1 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.)

5.39.2 Example

The ogrinfo utility can be used to dump the content of a GTM datafile :

```
ogrinfo -ro -al input.gtm
```

Use of -sql option to remap field names to the ones allowed by the GTM schema:

```
ogr2ogr -f "GPSTrackMaker" output.gtm input.shp -sql "SELECT field1 AS name, field2_
↳AS color, field3 AS type FROM input"
```

Example for translation from PostGIS to GTM:

```
ogr2ogr -f "GPSTrackMaker" output.gtm PG:"host=hostaddress user=username dbname=db_
↳password=mypassword" -sql "select filed1 as name, field2 as color, field3 as type,
↳wkb_geometry from input" -nlt MULTILINESTRING
```

Note : You need to specify the layer type as POINT, LINESTRING, or MULTILINESTRING.

5.39.3 See Also

- [Home page for GPS TrackMaker Program](#)
- [GTM 211 format documentation](#)

5.40 HTF - Hydrographic Transfer Format

Driver short name

HTF

Driver built-in by default

This driver is built-in by default

This driver reads files containing sounding data following the Hydrographic Transfer Format (HTF), which is used by the Australian Hydrographic Office (AHO).

The driver has been developed based on HTF 2.02 specification.

The file must be georeferenced in UTM WGS84 to be considered valid by the driver.

The driver returns 2 spatial layers : a layer named “polygon” and a layer name “sounding”. There is also a “hidden” layer, called “metadata”, that can be fetched with `GetLayerByName()`, and which contains a single feature, made of the header lines of the file.

Polygons are used to distinguish between differing survey categories, such that any significant changes in position/depth accuracy and/or a change in the seafloor coverage will dictate a separate bounding polygon contains polygons.

The “polygon” layer contains the following fields :

- *DESCRIPTION* : Defines the polygons of each region of similar survey criteria or theme.
- *IDENTIFIER* : Unique polygon identifier for this transmittal.
- *SEAFLOOR_COVERAGE* : All significant seafloor features detected (full ensonification/sweep) or full coverage not achieved and uncharted features may exist.
- *POSITION_ACCURACY* : +/- NNN.n meters at 95% CI (2.45) with respect to the given datum.
- *DEPTH_ACCURACY* : +/- NN.n meters at 95% CI (2.00) at critical depths.

The “sounding” layer should contain - at minimum - the following 20 fields :

- *REJECTED_SOUNDING* : if 0 sounding is valid or if 1 the sounding has been rejected (flagged).

- *LINE_NAME* : Survey line name/number as a unique identifier within the survey.
- *FIX_NUMBER* : Sequential sounding fix number, unique within the survey.
- *UTC_DATE* : UTC date for the sounding CCYYMMDD.
- *UTC_TIME* : UTC time for the sounding HHMMSS.ss.
- *LATITUDE* : Latitude position of the sounding +/-NN.nnnnnn (degrees of arc, south is negative).
- *LONGITUDE* : Longitude position of the sounding +/-NNN.nnnnnn (degrees of arc, west is negative).
- *EASTING* : Grid coordinate position of the sounding in meters NNNNNNN.n.
- *NORTHING* : Grid coordinate position of the sounding in meters NNNNNNN.n.
- *DEPTH* : Reduced sounding value in meters with corrections applied as indicated in the relevant fields, soundings are positive and drying heights are negative +/-NNNN.nn meters.
- *POSITIONING_SENSOR* : Indicate position system number populated in the HTF header record.
- *DEPTH_SENSOR* : Indicate depth sounder system number populated in the HTF header record.
- *TPE_POSITION* : Total propagated error of the horizontal component for the sounding.
- *TPE_DEPTH* : Total propagated error of the vertical component for the sounding.
- *NBA_FLAG* : No Bottom at Flag, if 0 not NBA depth or if 1 Depth is NBA, deeper water probably exists.
- *TIDE* : Value of the tidal correction applied +/- NN.nn meters.
- *DEEP_WATER_CORRECTION* : Value of the deep water sounding velocity applied +/- NN.nn meters.
- *VERTICAL_BIAS_CORRECTION* : Value of the vertical bias applied +/- NN.nn meters. eg transducer depth correction
- *SOUND_VELOCITY* : Measured sound velocity used to process sounding in meters per second III.
- *PLOTTED_SOUNDING* : if 0 then the reduced depth did not appear on the original fairsheet or id 1 then the reduced depth appeared on the original fairsheet.

Some fields may be never set, depending on the value of the Field Population Key. Extra fields may also be added.

5.40.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

5.40.2 See Also

- [HTF - Hydrographic Transfer Format home page](#)
- [HTF Technical Specification](#)

5.41 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.

5.41.1 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.)*

5.41.2 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
- **IDB_OGR_FID:** Set name of primary key instead of 'ogc_fid'.

For more information about Informix variables read documentation of Informix Client SDK

5.41.3 Example

This example shows using ogrinfo to list Informix DataBlade layers on a different host.

```
ogrinfo -ro IDB:"server=demo_on user=informix dbname=frames"
```

5.42 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.

5.42.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

5.43 “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 and writing.

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 2.3) 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

5.43.1 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.*)

5.43.2 Model support

Data is read and written into transfer files which have different formats in INTERLIS 1 (.itf) and INTERLIS 2 (.xtf). Models are passed in IlisMeta format by using “a_filename.xtf,models.imd” as a connection string.

IlisMeta files can 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 modell1.ili [modell2.ili ...]
```

Some possible transformations using *ogr2ogr*.

- Interlis 1 -> Shape:

```
ogr2ogr -f "ESRI Shapefile" shpdir ili-bsp.itf,Beispiel.imd
```

- Interlis 2 -> Shape:

```
ogr2ogr -f "ESRI Shapefile" shpdir RoadsExdm2ien.xml,RoadsExdm2ien.imd
```

or without model:

```
ogr2ogr -f "ESRI Shapefile" shpdir RoadsExdm2ien.xml
```

Example with curves and multiple geometries:

```
ogr2ogr --config OGR_STROKE_CURVE TRUE -SQL 'SELECT Rechtsstatus, publiziertAb,  
→MetadatenGeobasisdaten, Eigentumsbeschraenkung, ZustaendigeStelle, Flaechen FROM  
→"OeREBKRM09trsfr.Transferstruktur.Geometrie"' shpdir ch.bazl.  
→sicherheitszonenplan.oereb_20131118.xtf,OeREBKRM09vs.imd OeREBKRM09trsfr.  
→Transferstruktur.Geometrie
```

- Shape -> Interlis 2:

```
ogr2ogr -f "Interlis 2" LandCover.xml,RoadsExdm2ien.imd RoadsExdm2ben.Roads.  
→LandCover.shp
```

- Importing multiple Interlis 1 files into PostGIS:

```
ogr2ogr -f PostgreSQL PG:dbname=warmerda av_fixpunkte_ohne_LFPNachfuehrung.itf,av.
↳imd -lco OVERWRITE=yes
ogr2ogr -f PostgreSQL PG:dbname=warmerda av_fixpunkte_mit_LFPNachfuehrung.itf,av.
↳imd -append
```

5.43.2.1 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`.

5.43.3 Other Notes

- `ogrtools` library includes extensions for the OGR Interlis driver
- Development of the OGR INTERLIS driver was supported by [Swiss Federal Administration](#), [Canton Solothurn](#) and [Canton Thurgovia](#).

5.44 INGRES

Driver short name

INGRES

Build dependencies

INGRESS

This driver implements read and write access for spatial data in [INGRES](#) database tables. This functionality was introduced in GDAL/OGR 1.6.0.

When opening a database, its name should be specified in the form “@driver=ingres,[host=*host*, instance=*instance],dbname=[vnode::]dbname [options]”. where the options can include comma separated items like “host=*ip_address*”, “instance=*instance*”, “username=*userid*”, “password=*password*”, “effuser=*database_user*”, “dbpwd=*database_passwd*”, “timeout=*timeout*”, “tables=table1/table2”.

The driver and dbname values are required, while the rest are optional. If username and password are not provided an attempt is made to authenticate as the current OS user.

If the host and instance options are both specified, the username and password *must* be supplied as it creates a temporary [dynamic vnode connection](#). The default protocol is TCP/IP. If any other protocol is expected to be used, a pre-built vnode is preferable. If vnode and these two options are passed at the same time, an error will occur.

The option `effuser` and `dbpwd` are mapped to the real user name and password needs to be authorized in dbms, compared to the username and password which are used for OS level authorization.

Examples:

```
@driver=ingres,host=192.168.0.1, instance=II, dbname=test,user=warmerda,
↳password=test,effuser=frank, dbpwd=123, tables=usa/canada

@driver=ingres,host=192.168.0.1, instance=II, dbname=test,user=warmerda,
↳password=test,tables=usa/canada

@driver=ingres,dbname=test,user=warmerda,password=test,tables=usa/canada

@driver=ingres,dbname=test,user=warmerda,password=test,tables=usa/canada

@driver=ingres,dbname=test,user=warmerda,password=test,tables=usa/canada

@driver=ingres,dbname=server::mapping

@driver=ingres,dbname=mapping
```

If the tables list is not provided, an attempt is made to enumerate all non-system tables as layers, otherwise only the listed tables are represented as layers. 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.

If an integer field exists in a table that is named “ogr_fid” it will be used as the FID, otherwise FIDs will be assigned sequentially. This can result in different FIDs being assigned to a given record/feature depending on the spatial and attribute query filters in effect at a given time.

By default, SQL statements are passed directly to the INGRES 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 INGRES driver supports OGC SFSQL 1.1 compliant spatial types and functions, including types: POINT, LINESTRING, POLYGON, MULTI* versions, and GEOMETRYCOLLECTION.

5.44.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

5.44.2 Caveats

- No fast spatial index is used when reading, so spatial filters are implemented by reading and parsing all records, and then discarding those that do not satisfy the spatial filter.

5.44.3 Creation Issues

The INGRES driver does not support creation of new datasets (a database within INGRES), but it does allow creation of new layers (tables) within an existing database instance.

- The INGRES driver makes no allowances for character encodings at this time.
- The INGRES driver is not transactional at this time.

5.44.3.1 Layer Creation Options

- **OVERWRITE:** This may be “YES” to force an existing layer of the desired name to be destroyed before creating the requested layer.
- **LAUNDER:** 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. The default value is “YES”.
- **PRECISION:** This may be “TRUE” to attempt to preserve field widths and precisions for the creation and reading of MySQL layers. The default value is “TRUE”.
- **GEOMETRY_NAME:** This option specifies the name of the geometry column. The default value is “SHAPE”.
- **INGRES_FID:** This option specifies the name of the FID column. The default value is “OGR_FID”
- **GEOMETRY_TYPE:** Specifies the object type for the geometry column. It may be one of POINT, LSEG, LINE, LONG LINE, POLYGON, or LONG POLYGON. By default POINT, LONG LINE or LONG POLYGON are used depending on the layer type.

5.44.4 Older Versions

The INGRES GDAL driver also includes support for old INGRES spatial types, but these are not enabled by default. It enable these, the input *configure* script needs to include pointers to libraries used by the older version:

```
INGRES_LIB="-L$II_SYSTEM/ingres/lib \
    $II_SYSTEM/ingres/lib/iiclsadt.o \
    $II_SYSTEM/ingres/lib/iiuseradt.o \
    -liiapi.1 -lcompat.1 -lq.1 -lframe.1"
```

5.45 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.

5.45.1 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.*)

5.45.2 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.

5.45.3 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.

5.45.4 Creation Issues

The JML writer supports the following *layer* creation options:

- **CREATE_R_G_B_FIELD=**YES/NO: 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. Default value : YES
- **CREATE_OGR_STYLE_FIELD=**YES/NO: whether the create a OGR_STYLE field that will contain the Feature Style string. Default value : NO

5.45.5 See Also

- *Feature Style Specification*

5.45.6 Credits

The author wishes to thank Jukka Rahkonen for funding the development of this driver.

5.46 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)*

5.46.1 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.*)

5.46.1.1 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. Since GDAL 1.6.1, folders containing multiple geometry types, like `POINT` and `POLYGON`, are supported.

5.46.1.2 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 geo-browsers 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.

5.46.1.3 Encoding issues

Expat library supports reading the following built-in encodings :

- US-ASCII
- UTF-8
- UTF-16
- ISO-8859-1

OGR 1.8.0 adds supports for Windows-1252 encoding (for previous versions, altering the encoding mentioned in the XML header to ISO-8859-1 might work in some cases).

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.

5.46.1.4 Creation Options

The following dataset creation options are supported:

- **NameField**: Allows you to specify the field to use for the KML <name> element. Default value : ‘Name’
- **DescriptionField**: Allows you to specify the field to use for the KML <description> element. Default value : ‘Description’
- **AltitudeMode**: 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=string**: Starting with GDAL 2.2, the DOCUMENT_ID datasource creation option can be used to specified the id of the root <Document> node. The default value is root_doc.

5.46.2 VSI Virtual File System API support

(Some features below might require OGR >= 1.9.0)

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.

5.46.3 Example

The ogr2ogr utility can be used to dump 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'"
```

How to dump contents of .kml file as OGR sees it:

```
ogrinfo -ro somedisplay.kml
```

5.46.4 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.

5.46.5 See Also

- [KML Specification](#)
- [KML Tutorial](#)
- *[LIBKML driver](#)* An alternative GDAL KML driver

5.47 LIBKML Driver (.kml .kmz)

Driver short nameLIBKML

Build dependencieslibkml

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.

5.47.1 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.*)

5.47.2 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 environment variable `LIBKML_USE_DOC.KML` to “no”

5.47.2.1 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.

5.47.2.2 Datasource creation options

Starting with OGR 1.11, the following datasource creation options can be used to generate a `<atom:Author>` element at the top Document level.

- **AUTHOR_NAME**
- **AUTHOR_URI**
- **AUTHOR_EMAIL**

The href of an `<atom:link>` element at the top Document level can be specified with the **LINK** creation option.

The `<phoneNumber>` element at the top Document level can be specified with the **PHONENUMBER** creation option. The value must follow the syntax of [IETF RFC 3966](#).

Starting with GDAL 2.2, the `DOCUMENT_ID` datasource creation option can be used to specified the id of the root `<Document>` node. The default value is `root_doc`.

Container properties

The following dataset creation options can be used to set container options :

- **NAME**: `<name>` element
- **VISIBILITY**: `<visibility>` element
- **OPEN**: `<open>` element
- **SNIPPET**: `<snippet>` element
- **DESCRIPTION**: `<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, trough a `<ListStyle>` element:

- **LISTSTYLE_TYPE**: can be one of “check”, “radioFolder”, “checkOffOnly” or “checkHideChildren”. Sets the `<listItemType>` element.
- **LISTSTYLE_ICON_HREF**: 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` : to set the `<minRefreshPeriod>` element
- `NLC_MAXSESSIONLENGTH` : to set the `<maxSessionLength>` element
- `NLC_COOKIE` : to set the `<cookie>` element
- `NLC_MESSAGE` : to set the `<message>` element
- `NLC_LINKNAME` : to set the `<linkName>` element
- `NLC_LINKDESCRIPTION` : to set the `<linkDescription>` element
- `NLC_LINKSNIPPET` : to set the `<linkSnippet>` element
- `NLC_EXPIRES` : 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.

5.47.3 Layer

OGRLayer are mapped to kml files as a `<Document>` or `<Folder>`, and in kmz files or directories as a separate kml file.

5.47.3.1 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.

5.47.3.2 Schema

Read and write of `<Schema>` is supported for .kml files, .kmz files, and directories.

5.47.3.3 Layer creation options

Starting with OGR 1.11, the following layer creation options can be used to generate a `<LookAt>` element at the layer level.

- **LOOKAT_LONGITUDE** (required)
- **LOOKAT_LATITUDE** (required)
- **LOOKAT_RANGE** (required)
- **LOOKAT_HEADING**
- **LOOKAT_TILT**
- **LOOKAT_ALTITUDE**
- **LOOKAT_ALTITUDEMODE**

Alternatively, a `<Camera>` element can be generated.

- **CAMERA_LONGITUDE** (required)
- **CAMERA_LATITUDE** (required)
- **CAMERA_ALTITUDE** (required)
- **CAMERA_ALTITUDEMODE** (required)
- **CAMERA_HEADING**
- **CAMERA_TILT**
- **CAMERA_ROLL**

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** (optional) : defines the west coordinate of the region.
- **REGION_YMIN** (optional) : defines the south coordinate of the region.
- **REGION_XMAX** (optional) : defines the east coordinate of the region.
- **REGION_YMAX** (optional) : defines the north coordinate of the region.
- **REGION_MIN_LOD_PIXELS** (optional) : minimum size in pixels of the region so that it is displayed. Defaults to 256.
- **REGION_MAX_LOD_PIXELS** (optional) : maximum size in pixels of the region so that it is displayed. Defaults to -1 (infinite).
- **REGION_MIN_FADE_EXTENT** (optional) : distance over which the geometry fades, from fully opaque to fully transparent. Defaults to 0.
- **REGION_MAX_FADE_EXTENT** (optional) : distance over which the geometry fades, from fully transparent to fully opaque. Defaults to 0.

A `<ScreenOverlay>` element can be added to display a logo, a legend, etc. . .

- **SO_HREF** (required) : URL of the image to display.
- **SO_NAME** (optional)
- **SO_DESCRIPTION** (optional)
- **SO_OVERLAY_X** (optional)
- **SO_OVERLAY_Y** (optional)
- **SO_OVERLAY_XUNITS** (optional)
- **SO_OVERLAY_YUNITS** (optional)
- **SO_SCREEN_X** (optional). Defaults to 0.05
- **SO_SCREEN_Y** (optional). Defaults to 0.05
- **SO_SCREEN_XUNITS** (optional). Defaults to Fraction
- **SO_SCREEN_YUNITS** (optional). Defaults to Fraction
- **SO_SIZE_X** (optional)
- **SO_SIZE_Y** (optional)
- **SO_SIZE_XUNITS** (optional)
- **SO_SIZE_YUNITS** (optional)

By default, layers are written as `<Document>` elements. By settings the **FOLDER** layer creation 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**: `<name>` element
- **VISIBILITY**: `<visibility>` element
- **OPEN**: `<open>` element
- **SNIPPET**: `<snippet>` element
- **DESCRIPTION**: `<description>` element

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, trough a `<ListStyle>` element:

- **LISTSTYLE_TYPE**: can be one of “check”, “radioFolder”, “checkOffOnly” or “checkHideChildren”. Sets the `<listItemType>` element.
- **LISTSTYLE_ICON_HREF**: URL of the icon to display for the layer folder. Sets the href element of the `<ItemIcon>` element.

5.47.4 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.

Starting with OGR 1.10, 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.

5.47.4.1 Style

Style Strings at the feature level are Mapped to KML as either a [Style](#) or [StyleUrl](#) in each [Placemark](#).

When reading a kml feature and the environment variable LIBKML_RESOLVE_STYLE 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.

When reading a kml feature and the environment variable LIBKML_EXTERNAL_STYLE 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 is set to no then the styleurl is copied to the style string.

When reading a kml StyleMap the default mapping is set to normal. If you wish to use the highlighted styles set the environment variable LIBKML_STYLEMAP_KEY 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”.

5.47.5 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 if the LIBKML_USE_SCHEMADATA configuration option is set to NO.

A rich set of environment variables 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 an environment variable.

Name This String field maps to the kml tag [name](#). The name of the ogr field can be changed with the environment variable LIBKML_NAME_FIELD .

description This String field maps to the kml tag [description](#). The name of the ogr field can be changed with the environment variable LIBKML_DESCRIPTION_FIELD .

timestamp This string or datetime or date and/or time field maps to the kml tag [timestamp](#). The name of the ogr field can be changed with the environment variable LIBKML_TIMESTAMP_FIELD .

begin This string or datetime or date and/or time field maps to the kml tag [begin](#). The name of the ogr field can be changed with the environment variable LIBKML_BEGIN_FIELD .

end This string or datetime or date and/or time field maps to the kml tag [end](#). The name of the ogr field can be changed with the environment variable LIBKML_END_FIELD .

- altitudeMode** This string field maps to the kml tag `<altitudeMode>` or `<gx:altitudeMode>`. The name of the ogr field can be changed with the environment variable `LIBKML_ALTITUDEMODE_FIELD`.
- tessellate** This integer field maps to the kml tag `<tessellate>`. The name of the ogr field can be changed with the environment variable `LIBKML_TESSELLATE_FIELD`.
- extrude** This integer field maps to the kml tag `<extrude>`. The name of the ogr field can be changed with the environment variable `LIBKML_EXTRUDE_FIELD`.
- visibility** This integer field maps to the kml tag `<visibility>`. The name of the ogr field can be changed with the environment variable `LIBKML_VISIBILITY_FIELD`.
- icon** This string field maps to the kml tag `<icon>`. The name of the ogr field can be changed with the environment variable `LIBKML_ICON_FIELD`.
- drawOrder** This integer field maps to the kml tag `<drawOrder>`. The name of the ogr field can be changed with the environment variable `LIBKML_DRAWORDER_FIELD`.
- snippet** This integer field maps to the kml tag `<snippet>`. The name of the ogr field can be changed with the environment variable `LIBKML_SNIPPET_FIELD`.
- heading** This real field maps to the kml tag `<heading>`. The name of the ogr field can be changed with the environment variable `LIBKML_HEADING_FIELD`. When reading, this field is present only if a Placemark has a Camera with a heading element.
- tilt** This real field maps to the kml tag `<tilt>`. The name of the ogr field can be changed with the environment variable `LIBKML_TILT_FIELD`. When reading, this field is present only if a Placemark has a Camera with a tilt element.
- roll** This real field maps to the kml tag `<roll>`. The name of the ogr field can be changed with the environment variable `LIBKML_ROLL_FIELD`. When reading, this field is present only if a Placemark has a Camera with a roll element.
- model** This string field can be used to define the URL of a 3D `<model>`. The name of the ogr field can be changed with the environment variable `LIBKML_MODEL_FIELD`.
- scale_x** This real field maps to the x element of the kml tag `<scale>` for a 3D model. The name of the ogr field can be changed with the environment variable `LIBKML_SCALE_X_FIELD`.
- scale_y** This real field maps to the y element of the kml tag `<scale>` for a 3D model. The name of the ogr field can be changed with the environment variable `LIBKML_SCALE_Y_FIELD`.
- scale_z** This real field maps to the z element of the kml tag `<scale>` for a 3D model. The name of the ogr field can be changed with the environment variable `LIBKML_SCALE_Z_FIELD`.
- networklink** This string field maps to the href element of the kml tag `<href>` of a NetworkLink. The name of the ogr field can be changed with the environment variable `LIBKML_NETWORKLINK_FIELD`.
- networklink_refreshvisibility** This integer field maps to kml tag `<refreshVisibility>` of a NetworkLink. The name of the ogr field can be changed with the environment variable `LIBKML_NETWORKLINK_REFRESHVISIBILITY_FIELD`.
- networklink_flytoview** This integer field maps to kml tag `<flyToView>` of a NetworkLink. The name of the ogr field can be changed with the environment variable `LIBKML_NETWORKLINK_FLYTOVIEW_FIELD`.
- networklink_refreshmode** This string field maps to kml tag `<refreshMode>` of a NetworkLink. The name of the ogr field can be changed with the environment variable `LIBKML_NETWORKLINK_REFRESHMODE_FIELD`.
- networklink_refreshinterval** This real field maps to kml tag `<refreshInterval>` of a NetworkLink. The name of the ogr field can be changed with the environment variable `LIBKML_NETWORKLINK_REFRESHINTERVAL_FIELD`.

- networklink_viewrefreshmode** This string field maps to kml tag `<viewRefreshMode>` of a `NetworkLink`. The name of the ogr field can be changed with the environment variable `LIBKML_NETWORKLINK_VIEWREFRESHMODE_FIELD`.
- networklink_viewrefreshtime** This real field maps to kml tag `<viewRefreshTime>` of a `NetworkLink`. The name of the ogr field can be changed with the environment variable `LIBKML_NETWORKLINK_VIEWREFRESHTIME_FIELD`.
- networklink_viewboundscale** This real field maps to kml tag `<viewBoundScale>` of a `NetworkLink`. The name of the ogr field can be changed with the environment variable `LIBKML_NETWORKLINK_VIEWBOUNDSCALE_FIELD`.
- networklink_viewformat** This string field maps to kml tag `<viewFormat>` of a `NetworkLink`. The name of the ogr field can be changed with the environment variable `LIBKML_NETWORKLINK_VIEWFORMAT_FIELD`.
- networklink_httpquery** This string field maps to kml tag `<httpQuery>` of a `NetworkLink`. The name of the ogr field can be changed with the environment variable `LIBKML_NETWORKLINK_HTTPQUERY_FIELD`.
- camera_longitude** This real field maps to kml tag `<longitude>` of a `<Camera>`. The name of the ogr field can be changed with the environment variable `LIBKML_CACameraMERA_LONGITUDE_FIELD`.
- camera_latitude** This real field maps to kml tag `<latitude>` of a `<Camera>`. The name of the ogr field can be changed with the environment variable `LIBKML_CAMERA_LATITUDE_FIELD`.
- camera_altitude** This real field maps to kml tag `<altitude>` of a `<Camera>`. The name of the ogr field can be changed with the environment variable `LIBKML_CAMERA_ALTITUDE_FIELD`.
- camera_altitudemode** This real field maps to kml tag `<altitudeMode>` of a `<Camera>`. The name of the ogr field can be changed with the environment variable `LIBKML_CAMERA_ALTITUDEMODOE_FIELD`.
- photooverlay** This string field maps to the href element of the kml tag `<href>` of a `<PhotoOverlay>`. The name of the ogr field can be changed with the environment variable `LIBKML_PHOTOOVERLAY_FIELD`.
- leftfov** This real field maps to to kml tag `<LeftFov>` of a `<PhotoOverlay>`. The name of the ogr field can be changed with the environment variable `LIBKML_LEFTFOV_FIELD`.
- rightfov** This real field maps to to kml tag `<RightFov>` of a `<PhotoOverlay>`. The name of the ogr field can be changed with the environment variable `LIBKML_RightFOV_FIELD`.
- bottomfov** This real field maps to to kml tag `<BottomFov>` of a `<PhotoOverlay>`. The name of the ogr field can be changed with the environment variable `LIBKML_BOTTOMTFOV_FIELD`.
- topfov** This real field maps to to kml tag `<TopFov>` of a `<PhotoOverlay>`. The name of the ogr field can be changed with the environment variable `LIBKML_TOPFOV_FIELD`.
- near** This real field maps to to kml tag `<Near>` of a `<PhotoOverlay>`. The name of the ogr field can be changed with the environment variable `LIBKML_NEAR_FIELD`.
- shape** This string field maps to to kml tag `<shape>` of a `<PhotoOverlay>`. The name of the ogr field can be changed with the environment variable `LIBKML_SHAPE_FIELD`.
- imagepyramid_tilesize** This integer field maps to to kml tag `<tileSize>` of a `<ImagePyramid>`. The name of the ogr field can be changed with the environment variable `LIBKML_IMAGEPYRAMID_TILESIZE`.
- imagepyramid_maxwidth** This integer field maps to to kml tag `<maxWidth>` of a `<ImagePyramid>`. The name of the ogr field can be changed with the environment variable `LIBKML_IMAGEPYRAMID_MAXWIDTH`.
- imagepyramid_maxheight** This integer field maps to to kml tag `<maxHeight>` of a `<ImagePyramid>`. The name of the ogr field can be changed with the environment variable `LIBKML_IMAGEPYRAMID_MAXHEIGHT`.
- imagepyramid_gridorigin** This string field maps to to kml tag `<gridOrigin>` of a `<ImagePyramid>`. The name of the ogr field can be changed with the environment variable `LIBKML_IMAGEPYRAMID_GRIDORIGIN`.

OGR_STYLE This string field maps to a features style string, OGR reads this field if there is no style string set on the feature.

5.47.6 Geometry

Translation of *OGRGeometry* to KML Geometry is pretty strait forwards 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.

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 the environment variable `LIBKML_WRAPDATELINE` to “yes” will cause the libkml driver to split the geometry at the dateline when read.

5.47.7 VSI Virtual File System API support

(Some features below might require OGR >= 1.9.0)

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) , `/vsizip/` (read-write) , `/vsicurl/` (read-only) domains.

Writing to `/dev/stdout` or `/vsistdout/` is also supported.

5.47.8 Example

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.
```

(continues on next page)

(continued from previous page)

```

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=', '

    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
}

```

(continues on next page)

(continued from previous page)

```
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

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
```

5.48 MapML

New 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.

5.48.1 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.*)

5.48.2 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.

5.48.3 Write support

Several layers can be written in the same MapML file.

Only the following CRS are natively supported, 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.

5.48.4 Dataset creation options

- **HEAD**=string: Filename or inline XML content for head element.
- **EXTENT_UNITS**=AUTO/WGS84/OSMTILE/CBMTILE/APSTILE: To override the CRS.
- **EXTENT_ACTION**=string: Value of `extent@action` attribute.
- **EXTENT_XMIN**=float: Override extent xmin value.
- **EXTENT_YMIN**=float: Override extent ymin value.
- **EXTENT_XMAX**=float: Override extent xmax value.
- **EXTENT_YMAX**=float: Override extent ymax value.
- **EXTENT_XMIN_MIN**=float: Min value for extent.xmin value.
- **EXTENT_XMIN_MAX**=float: Max value for extent.xmin value.

- **EXTENT_YMIN_MIN**=float: Min value for extent.ymin value.
- **EXTENT_YMIN_MAX**=float: Max value for extent.ymin value.
- **EXTENT_XMAX_MIN**=float: Min value for extent.xmax value.
- **EXTENT_XMAX_MAX**=float: Max value for extent.xmax value.
- **EXTENT_YMAX_MIN**=float: Min value for extent.ymax value.
- **EXTENT_YMAX_MAX**=float: Max value for extent.ymax value.
- **EXTENT_ZOOM**=int: Value of extent.zoom.
- **EXTENT_ZOOM_MIN**=int: Min value for extent.zoom.
- **EXTENT_ZOOM_MAX**=int: Max value for extent.zoom.
- **EXTENT_EXTRA**=string: Filename of inline XML content for extra content to insert in extent element.
- **BODY_LINKS**=string: Inline XML content for extra content to insert as link elements in the body. For example `'<link type="foo" href="bar" /><link type="baz" href="baw" />'`

5.48.5 Links

- [MapML specification](#)
- [MapML schemas](#)

5.49 Access MDB databases

Driver short name

MDB

Build dependencies

JDK/JRE and Jackcess

OGR optionally supports reading access .mdb files by using the Java [Jackcess](#) library.

This driver is primarily meant as being used on Unix platforms to overcome the issues often met with the MDBTools library that acts as the ODBC driver for MDB databases.

The driver can detect ESRI Personal Geodatabases and Geomedia MDB databases, and will deal them exactly as the *PGeo* and *Geomedia* drivers do. For other MDB databases, all the tables will be presented as OGR layers.

5.49.1 How to build the MDB driver (on Linux)

You need a JDK (a JRE is not enough) to build the driver. On Ubuntu 10.04 with the `openjdk-6-jdk` package installed,

```
./configure --with-java=yes --with-mdb=yes
```

On others Linux flavors, you may need to specify :

```
./configure --with-java=/path/to/jdk/root/path --with-jvm-lib=/path/to/libjvm/  
↳directory --with-mdb=yes
```

where `/path/to/libjvm/directory` is for example `/usr/lib/jvm/java-6-openjdk/jre/lib/amd64/server`

It is possible to add the `--with-jvm-lib-add-rpath` option (no value or “yes”) to embed the path to the `libjvm.so` in the GDAL library.

5.49.2 How to run the MDB driver (on Linux)

You need a JRE and 3 external JARs to run the driver.

1. If you didn't specify `--with-jvm-lib-add-rpath` at configure time, set the path of the directory that contains `libjvm.so` in `LD_LIBRARY_PATH` or in `/etc/ld.so.conf`.
2. Download `jackcess-1.2.XX.jar` (but 2.X does not work with the current driver), `commons-lang-2.4.jar` and `commons-logging-1.1.1.jar` (other versions might work)
3. Put the 3 JARs either in the `lib/ext` directory of the JRE (e.g. `/usr/lib/jvm/java-6-openjdk/jre/lib/ext`) or in another directory and explicitly point to each of them with the `CLASSPATH` environment variable.

5.49.3 Resources

- [Jackcess](#) library home page
- Utility that contains the needed JARs dependencies

5.49.4 See also

- [PGeo](#) driver page
- [Geomedia](#) driver page

5.50 Memory

Driver short name

Memory

Driver built-in by default

This driver is built-in by default

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 Memory datastore. It must be created with `CreateDataSource()` and populated and used from that handle. When the datastore 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).

5.50.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

5.50.2 Creation Issues

Any name may be used for a created datasource. There are no datasource or layer creation options supported. Layer names need to be unique, but are not otherwise constrained.

Before GDAL 2.1, feature ids passed to `CreateFeature()` are preserved *unless* they exceed 10000000 in which case they will be reset to avoid a requirement for an excessively large and sparse feature array. Starting with GDAL 2.1, sparse IDs can be handled.

New fields can be added or removed to a layer that already has features.

5.51 MapInfo TAB and MIF/MID

Driver short name

MITAB

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. Starting with GDAL 2.0, 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.

5.51.1 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.*)

5.51.2 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)

Starting with GDAL 2.0, 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, 49.000000000002, 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, 49.000000000001, ↵
↵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.

5.51.2.1 Dataset Creation Options

- **FORMAT=MIF**: To create MIF/MID instead of TAB files (TAB is the default).
- **SPATIAL_INDEX_MODE=QUICK/OPTIMIZED**: The default is QUICK force “quick spatial index mode”. In this 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 in GDAL 2.0 to generate optimized spatial index.
- **BLOCKSIZE=[512,1024,...,32256]** (multiples of 512): (GDAL >= 2.0.2) Block size for .map files. Defaults to 512. GDAL 2.0.1 and earlier versions only supported BLOCKSIZE=512 in reading and writing. 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.

5.51.2.2 Layer Creation Options

- **BOUNDSEXMIN,YMIN,XMAX,YMAX**: (GDAL >=2.0) 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**: (GDAL >=2.3) 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 ‘’ that equals to ‘Neutral’ MapInfo charset.
- **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.

5.51.2.3 Configuration options

- `MITAB_SET_TOWGS84_ON_KNOWN_DATUM=YES/NO`: (GDAL \geq 3.0.3). The default behaviour, 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.

5.51.2.4 See Also

- [MITAB Page](#)
- [About friendly layer names](#)

5.52 MongoDB

New in version 2.1.

Driver short name

MongoDB

Build dependencies

Mongo C++ client legacy 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 uses the legacy MongoDB C++ driver client library. To connect to MongoDB 3.0 or later servers, starting with GDAL 3.0, use the new [MongoDBv3](#) driver which uses the MongoDB C++ v3.4.0 client library. This driver will be eventually in favor of MongoDBv3

5.52.1 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.*)

5.52.2 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.

5.52.3 Dataset name syntax

There are two main possible syntaxes:

- One using [MongoDB URI](#), such as `mongodb://[usr:pwd@]host1[:port1]...[hostN[:portN]]/[db][?options]`
- One using just MongoDB: as the name and open options to specify host, port, user, password, database, etc. . .

The open options available are :

- **URI**=uri: [Connection URI](#)
- **HOST**=hostname: Server hostname. Default to localhost.
- **PORT**=port. Server port. Default to 27017.
- **DBNAME**=dbname. Database name. Should be specified when connecting to hosts with user authentication enabled.
- **AUTH_DBNAME**=dbname. Authentication database name, in case it is different from the database to work onto.
- **USER**=name. User name.
- **PASSWORD**=password. User password.
- **AUTH_JSON**=json_string. Authentication elements as JSON object. This is for advanced authentication. The JSON fields to put in the dictionary are :
 - “mechanism”: The string name of the sasl mechanism to use (MONGODB-CR, SCRAM-SHA-1 or DEFAULT). Mandatory.
 - “user”: The string name of the user to authenticate. Mandatory.
 - “db”: The database target of the auth command, which identifies the location of the credential information for the user. May be “\$external” if credential information is stored outside of the mongo cluster. Mandatory.
 - “pwd”: The password data.
 - “digestPassword”: Boolean, set to true if the “pwd” is undigested (default)
 - “pwd”: The password data.
 - “serviceHostname”: The GSSAPI hostname to use. Defaults to the name of the remote host.
- **SSL_PEM_KEY_FILE**=filename. SSL PEM certificate/key filename.
- **SSL_PEM_KEY_PASSWORD**=password. 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. Whether to allow connections to servers with invalid certificates. Defaults to NO.
- **SSL_ALLOW_INVALID_HOSTNAMES**=YES/NO. Whether to allow connections to servers with non-matching hostnames. Defaults to NO.

- **FIPS_MODE=YES/NO.** Whether to activate FIPS 140-2 mode at startup. Defaults to NO.
- **BATCH_SIZE=number.** 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=number.** Number of features to retrieve to establish feature definition. -1 = unlimited. Defaults to 100.
- **JSON_FIELD=YES/NO.** Whether to include a field called “_json” with the full document as JSON. Defaults to NO.
- **FLATTEN_NESTED_ATTRIBUTE=YES/NO.** Whether to recursively explore nested objects and produce flatten OGR attributes. Defaults to YES.
- **FID=name.** Field name, with integer values, to use as FID. Defaults to ogc_fid.
- **USE_OGR_METADATA=YES/NO.** Whether to use the _ogr_metadata collection to read layer metadata. Defaults to YES.
- **BULK_INSERT=YES/NO.** Whether to use bulk insert for feature creation. Defaults to YES.

Note: the `SSL_*` and `FIPS_MODE` options must be set to the same values when opening multiple types MongoDB databases. This is a limitation of the Mongo C++ driver.

5.52.4 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](#).

5.52.5 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.

5.52.6 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).

5.52.7 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.

5.52.8 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.

5.52.9 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.

5.52.10 Layer creation options

The following layer creation options are supported:

- **OVERWRITE=**YES/NO. Whether to overwrite an existing collection with the layer name to be created. Defaults to NO.
- **GEOMETRY_NAME**=name. Name of geometry column. Defaults to `'geometry'`.
- **SPATIAL_INDEX**=YES/NO. Whether to create a spatial index (2dsphere). Defaults to YES.
- **FID**=string. Field name, with integer values, to use as `FID`. Defaults to `'ogc_fid'`
- **WRITE_OGR_METADATA**=YES/NO. Whether to create a description of layer fields in the `_ogr_metadata` collection. Defaults to YES.
- **DOT_AS_NESTED_FIELD**=YES/NO. Whether to consider dot character in field name as sub-document. Defaults to YES.
- **IGNORE_SOURCE_ID**=YES/NO. Whether to ignore `_id` field in features passed to `CreateFeature()`. Defaults to NO.

5.52.11 Examples

Listing the tables of a MongoDB database:

```
ogrinfo -ro mongodb://user:password@ds047612.mongolab.com:47612/gdalautotest
```

Filtering on a MongoDB field:

```
ogrinfo -ro mongodb://user:password@ds047612.mongolab.com:47612/gdalautotest -where '
↳ { "field": 5 }'
```

Creating and populating a collection from a shapefile:

```
ogr2ogr -update mongodb://user:password@ds047612.mongolab.com:47612/gdalautotest_
↳ shapefile.shp
```

5.52.12 Build instructions

GDAL/OGR must be built against the [MongoDB C++ driver client library](#), in its “legacy” version (tested with 1.0.2), in order to the MongoDB driver to be compiled.

You must first follow [MongoDB C++ driver client build instructions](#), which require to have Boost libraries available.

Then:

- On Linux/Unix, run `./configure --with-mongocxx=/path/to/installation/root` (if the driver is already installed in `/usr`, this is not needed). If the Boost libraries are not found in the system paths, the path to the directory when the libraries are found can be specified `--with-boost-lib-path=/path/to/boost/libs`.
- On Windows, uncomment and adapt the following in `nmake.opt` (or add in `nmake.local`):

```
# Uncomment for MongoDB support
# This configuration is valid for a libmongoclient built as a DLL with:
# scons.bat --32 --dynamic-windows --sharedclient --prefix=c:\users\even\dev\
↳ mongo-client-install
#
#       --cpppath=c:\users\even\dev\boost_1_55_0_32bit --libpath=c:\users\
↳ even\dev\boost_1_55_0_32bit\lib32-msvc-10.0 install

# Uncomment if plugin is preferred
#MONGODB_PLUGIN = YES

MONGODB_INC = c:/users/even/dev/mongo-client-install/include
# Boost library names must be edited to reflect the actual MSVC and Boost versions
BOOST_INC = c:/users/even/dev/boost_1_55_0_32bit
BOOST_LIB_PATH= c:\users\even\dev\boost_1_55_0_32bit\lib32-msvc-10.0
MONGODB_LIBS = c:/users/even/dev/mongo-client-install/lib/mongoclient.lib \
               $(BOOST_LIB_PATH)\libboost_thread-vc100-mt-1_55.lib \
               $(BOOST_LIB_PATH)\libboost_system-vc100-mt-1_55.lib \
               $(BOOST_LIB_PATH)\libboost_date_time-vc100-mt-1_55.lib \
               $(BOOST_LIB_PATH)\libboost_chrono-vc100-mt-1_55.lib
```

5.52.13 See Also

- [MongoDB C++ Driver](#)
- [MongoDB 2.6 Manual](#)

5.53 MongoDBv3

New 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.

5.53.1 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.*)

5.53.2 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.

5.53.3 Dataset name syntax

There are two main possible syntaxes:

- One using **MongoDB URI** prefixed with **MONGODBV3:**, such as **MON-GODBV3: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 *MongoDB* driver. If the URI is starting with *mongodb+srv://*, then it is not needed.

The open options available are :

- **URI=uri:** **Connection URI**
- **HOST=hostname:** Server hostname. Default to localhost.
- **PORT=port.** Server port. Default to 27017.
- **DBNAME=dbname.** Database name. Should be specified when connecting to hosts with user authentication enabled.
- **USER=name.** User name.
- **PASSWORD=password.** User password.
- **SSL_PEM_KEY_FILE=filename.** SSL PEM certificate/key filename.
- **SSL_PEM_KEY_PASSWORD=password.** 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.** Whether to allow connections to servers with invalid certificates. Defaults to NO.
- **BATCH_SIZE=number.** 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=number.** Number of features to retrieve to establish feature definition. -1 = unlimited. Defaults to 100.
- **JSON_FIELD=YES/NO.** Whether to include a field called “_json” with the full document as JSON. Defaults to NO.
- **FLATTEN_NESTED_ATTRIBUTE=YES/NO.** Whether to recursively explore nested objects and produce flatten OGR attributes. Defaults to YES.
- **FID=name.** Field name, with integer values, to use as FID. Defaults to *ogc_fid*.
- **USE_OGR_METADATA=YES/NO.** Whether to use the *_ogr_metadata* collection to read layer metadata. Defaults to YES.
- **BULK_INSERT=YES/NO.** Whether to use bulk insert for feature creation. Defaults to YES.

5.53.4 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](#).

5.53.5 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.

5.53.6 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).

5.53.7 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.

5.53.8 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.

5.53.9 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.

5.53.10 Layer creation options

The following layer creation options are supported:

- **OVERWRITE=**YES/NO. Whether to overwrite an existing collection with the layer name to be created. Defaults to NO.
- **GEOMETRY_NAME=**name. Name of geometry column. Defaults to 'geometry'.
- **SPATIAL_INDEX=**YES/NO. Whether to create a spatial index (2dsphere). Defaults to YES.
- **FID=**string. Field name, with integer values, to use as FID. Defaults to 'ogc_fid'
- **WRITE_OGR_METADATA=**YES/NO. Whether to create a description of layer fields in the `_ogr_metadata` collection. Defaults to YES.
- **DOT_AS_NESTED_FIELD=**YES/NO. Whether to consider dot character in field name as sub-document. Defaults to YES.
- **IGNORE_SOURCE_ID=**YES/NO. Whether to ignore `_id` field in features passed to `CreateFeature()`. Defaults to NO.

5.53.11 Examples

Listing the tables of a MongoDB database:

```
ogrinfo -ro mongodb+srv://user:password@cluster0-ox9uy.mongodb.net/test
```

Filtering on a MongoDB field:

```
ogrinfo -ro mongodb+srv://user:password@cluster0-ox9uy.mongodb.net/test -where '{
↪ "field": 5 }'
```

Creating and populating a collection from a shapefile:

```
ogr2ogr -update mongodb+srv://user:password@cluster0-ox9uy.mongodb.net/test shapefile.
↪ shp
```

5.53.12 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:/
↪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
```

5.53.13 See Also

- [MongoDB C++ Driver](#)
- [MongoDB Manual](#)

5.54 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.

5.54.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

5.54.2 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**}

5.54.3 Layers

Starting with GDAL 1.11 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 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. Starting with GDAL 1.10 if the user defines the environment variable *MSSQLSPATIAL_USE_GEOMETRY_COLUMNS=NO* then the driver will look for all user spatial tables found in the system catalog

5.54.4 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.

5.54.5 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.

5.54.5.1 Layer Creation Options

- **GEOM_TYPE**: 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**: This may be “YES” to force an existing layer of the desired name to be destroyed before creating the requested layer.
- **LAUNDER**: 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. The default value is “YES”. If enabled the table (layer) name will also be laundered.
- **PRECISION**: 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. The default is “YES”.
- **DIM={2,3}**: Control the dimension of the layer. Defaults to 3.
- **GEOMETRY_NAME**: Set the name of geometry column in the new table. If omitted it defaults to *ogr_geometry*.. Note: option was called GEOM_NAME in releases before GDAL 2
- **SCHEMA**: Set name of schema for new table. If this parameter is not supported the default schema “dbo” is used.
- **SRID**: 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**: (From GDAL 2.0.0) Boolean flag (YES/NO) to enable/disable the automatic creation of a spatial index on the newly created layers (enabled by default).
- **UPLOAD_GEOM_FORMAT**: (From GDAL 2.0.0) Specify the geometry format (wkb or wkt) when creating or modifying features. The default is wkb.
- **FID**: (From GDAL 2.0.0) Name of the FID column to create. Defaults to *ogr_fid*.
- **FID64**: (From GDAL 2.0.0) Specifies whether to create the FID column with bigint type to handle 64bit wide ids. Default = NO
- **GEOMETRY_NULLABLE**: (From GDAL 2.0.0) Specifies whether the values of the geometry column can be NULL. Default = YES
- **EXTRACT_SCHEMA_FROM_LAYER_NAME**: (From GDAL 2.3.0) Can be set to NO to avoid considering the dot character as the separator between the schema and the table name. Defaults to YES.

5.54.5.2 Spatial Index Creation

By default the MS SQL Spatial driver doesn't add spatial indexes to the tables during the layer creation. However you should create a spatial index by using the following sql option:

```
create spatial index on schema.table
```

The spatial index can also be dropped by using the following syntax:

```
drop spatial index on schema.table
```

5.54.6 Configuration options

There are a variety of [Configuration Options](#) which help control the behavior of this driver.

- **MSSQLSPATIAL_USE_BCP:** (From GDAL 2.1.0) Enable bulk insert when adding features. This option requires to compile GDAL against a bulk copy enabled ODBC driver like SQL Server Native Client 11.0. To specify a BCP supported driver in the connection string, use the driver parameter, like DRIVER={SQL Server Native Client 11.0}. If GDAL is compiled against SQL Server Native Client 10.0 or 11.0 the driver is selected automatically not requiring to specify that in the connection string. If GDAL is compiled against SQL Server Native Client 10.0 or 11.0 the default setting of this parameter is TRUE, otherwise the parameter is ignored by the driver.
- **MSSQLSPATIAL_BCP_SIZE:** (From GDAL 2.1.0) Specifies the bulk insert batch size. The larger value makes the insert faster, but consumes more memory. Default = 1000.
- **MSSQLSPATIAL_OGR_FID:** Override FID column name. Default = ogr_fid.
- **MSSQLSPATIAL_ALWAYS_OUTPUT_FID:** Always retrieve the FID value of the recently created feature (even if it is not a true IDENTITY column). Default = "NO".
- **MSSQLSPATIAL_SHOW_FID_COLUMN:** Force to display the FID columns as a feature attribute. Default = "NO".
- **MSSQLSPATIAL_USE_GEOMETRY_COLUMNS:** Use/create geometry_columns metadata table in the database. Default = "YES".
- **MSSQLSPATIAL_LIST_ALL_TABLES:** Use mssql catalog to list available layers. Default = "NO".
- **MSSQLSPATIAL_USE_GEOMETRY_VALIDATION:** (From 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. Default = "YES".

5.54.7 Transaction support (GDAL >= 2.0)

The driver implements transactions at the dataset level, per rfc-54

5.54.8 Examples

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"
```

Connecting to a layer and dump 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"
```

Creating a spatial index

```
ogrinfo -sql "create spatial index on rivers" "MSSQL:server=.\br/>↳MSSQLSERVER2008;database=geodb;trusted_connection=yes"
```

Connecting with username/password

```
ogrinfo -al MSSQL:server=.\MSSQLSERVER2008;database=geodb;trusted_  
↳connection=no;UID=user;PWD=pwd
```

5.55 MVT: Mapbox Vector Tiles

New in version 2.3.

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.

5.55.1 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.*)

5.55.2 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.

5.55.3 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,
      \"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 and *tile_dimension_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,
}
```

5.55.4 Opening options

The following open options are available:

- **X=int_value**: X coordinate of the EPSG:3857 tile.
- **Y=int_value**: Y coordinate of the EPSG:3857 tile.
- **Z=int_value**: 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}.pbif` 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**: 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=string**: For tilesets, extension of tiles. Defaults to pbif.
- **TILE_COUNT_TO_ESTABLISH_FEATURE_DEFN=int_value**: For tilesets without metadata file, maximum number of tiles to use to establish the layer schemas. Defaults to 1000.

5.55.5 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.

5.55.6 Dataset creation options

- **NAME=string**: Tileset name. Defaults to the basename of the output file/directory. Used to fill metadata records.
- **DESCRIPTION=string**: 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}.pbif`. `MBTILES` is for a MBTILES container. Defaults to `DIRECTORY`, unless the output filename has a `.mbtiles` extension
- **TILE_EXTENSION=string**: For tilesets as directories of files, extension of tiles. Defaults to pbif.

- **MINZOOM**=integer: Minimum zoom level at which tiles are generated. Defaults to 0.
- **MAXZOOM**=integer: Maximum zoom level at which tiles are generated. Defaults to 5. Maximum supported value is 22
- **CONF**=string: 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. Number of units in a tile. The greater, the more accurate geometry coordinates (at the expense of tile byte size). Defaults to 4096
- **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. Whether to compress tiles with the Deflate/GZip algorithm. Defaults to YES. Should be left to YES for **FORMAT**=MBTILES.
- **TEMPORARY_DB**=string. 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. Maximum size of a tile in bytes (after compression). Defaults to 500 000. If a tile is greater than this threshold, features will be written with reduced precision, or discarded.
- **MAX_FEATURES**=integer. Maximum number of features per tile. Defaults to 200 000.
- **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: 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.

5.55.7 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,
```

(continues on next page)

(continued from previous page)

```

        "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 behaviour with the below layer creation options, although that is not convenient in the ogr2ogr use case.

5.55.8 Layer creation options

- **MINZOOM**=integer: Minimum zoom level at which tiles are generated. Defaults to the dataset creation option MINZOOM value.
- **MAXZOOM**=integer: Minimum zoom level at which tiles are generated. Defaults to the dataset creation option MAXZOOM value. Maximum supported value is 22
- **NAME**=string: 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**=string: A description of the layer.

5.55.9 Examples

```
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}"
```

```
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](#)

5.56 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.

5.56.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

5.56.2 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.

5.56.3 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.

5.56.3.1 Layer Creation Options

- **OVERWRITE**: This may be “YES” to force an existing layer of the desired name to be destroyed before creating the requested layer.
- **LAUNDER**: 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. The default value is “YES”.
- **PRECISION**: This may be “TRUE” to attempt to preserve field widths and precisions for the creation and reading of MySQL layers. The default value is “TRUE”.
- **GEOMETRY_NAME**: This option specifies the name of the geometry column. The default value is “SHAPE”.
- **FID**: This option specifies the name of the FID column. The default value is “OGR_FID”. Note: option was called `MYSQL_FID` in releases before GDAL 2
- **FID64**: (GDAL >= 2.0) This may be “TRUE” to create a FID column that can support 64 bit identifiers. The default value is “FALSE”.
- **SPATIAL_INDEX**: May be “NO” to stop automatic creation of a spatial index on the geometry column, allowing NULL geometries and possibly faster loading.

- **ENGINE:** Optionally specify database engine to use. In MySQL 4.x this must be set to MyISAM for spatial tables.

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
```

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
```

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)
```

5.57 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 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.

In GDAL 2.3 a bunch of workarounds were removed, that caused the driver to remap or ignore some elements and attributes internally to avoid attribute conflicts (e.g. *zeigtAufExternes*). Instead it now takes the **NAS_GFS_TEMPLATE** option, that 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*).

A [GFS template](#) and a corresponding [PostgreSQL schema](#) of the full NAS schema are part of [norGIS ALKIS-Import](#) (also featuring a shell script and PyQt frontend which ease the import). The two files were generated using [xmi2db](#) (fork of [xmi2db](#)) from the official application schema.

New in 2.3 is also the option **NAS_NO_RELATION_LAYER** that allows to disable populating the table *alkis_beziehungen*. The information found there is redundant to the relation fields also contained in original elements/tables. Enabling the option also makes progress reporting available.

This driver was implemented within the context of the [PostNAS project](#), which has more information on its use and other related projects.

5.57.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

5.58 NetCDF: Network Common Data Form - Vector

New in version 2.1.

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” and “profile” [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.

5.58.1 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.*)

5.58.2 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.

5.58.2.1 Distinguishing the Two Formats

Upon reading a netCDF file, the driver will attempt to read the global *Conventions* attribute. If its value is *CF-1.8* or higher (in this exact format, as specified in the CF convention) then the driver will treat the netCDF file as one that has *CF-1.8* geometries contained within it. If the *Conventions* attribute has a value of CF-1.6, the file will be treated as following the CF-1.6 convention.

5.58.2.2 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.

5.58.3 Mapping of concepts

5.58.3.1 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

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

5.58.3.2 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.

5.58.3.3 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.

5.58.3.4 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.

5.58.3.5 "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 exists).

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.

5.58.4 Dataset creation options

- **GEOMETRY_ENCODING=CF_1.8/WKT**: Chooses which geometry encoding to use when creating new layers within the dataset. Default is CF_1.8.
- **FORMAT=NC/NC2/NC4/NC4C**: netCDF format. NC is the classic netCDF format (compatible of 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. Default is NC.
- **WRITE_GDAL_TAGS=YES/NO**: Whether to write GDAL specific information as netCDF attributes. Default is YES.
- **CONFIG_FILE=string**. 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**. 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.

5.58.5 Layer creation options

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=string**. Name of the unlimited dimension that index features. Defaults to “record”.
- **STRING_DEFAULT_WIDTH=int**. Default width of strings (when using bi-dimensional char variables). Default is 10 in autogrow mode, 80 otherwise.
- **WKT_DEFAULT_WIDTH=int**. Default width of WKT strings (when using bi-dimensional char variables). Default is 1000 in autogrow mode, 10000 otherwise.
- **AUTOGROW_STRINGS=YES/NO**. Whether to auto-grow string fields of non-fixed width, or ogc_wkt special field, when serialized as bidimensional char variables. Default is YES. 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**. 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=string**. Name of the profile dimension and variable. Defaults to “profile”. Only used when FEATURE_TYPE=PROFILE.
- **PROFILE_DIM_INIT_SIZE=int or string**. 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=string**. 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**=int. 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. 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. The default value is NO.

5.58.6 XML configuration file

A XML configuration file conforming to the following [schema](#) 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 -->
```

(continues on next page)

(continued from previous page)

```

    <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

5.58.7 Further Reading

- *Raster side of the netCDF driver.*
- NetCDF CF-1.6 convention
- NetCDF CF-1.8 convention draft
- NetCDF compiled libraries
- NetCDF Documentation

5.58.8 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.

5.59 NGW – NextGIS Web

New in version 2.4.

Driver short name

NGW

Build dependencies

libcurl

NextGIS Web - is a server GIS, which allows to store and edit 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](#)).

5.59.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

5.59.2 Driver

The driver can connect to the services implementing the NextGIS Web REST API. NGW driver requires cURL support in GDAL. The driver supports read and write operations.

5.59.3 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.

5.59.4 Configuration options

The following configuration options are available:

- **NGW_USERPWD**: User name and password separated with colon. Optional and can be set using open options.
- **NGW_BATCH_SIZE**: 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**: 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**: Whether to store the json *extensions* key in feature native data.
- **NGW_JSON_DEPTH**: The depth of json response that can be parsed. If depth is greater than this value, parse error occurs.

5.59.5 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.

5.59.6 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](#)

5.59.7 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.

5.59.8 Field data types

NextWeb supports only following field types:

- OFTInteger
- OFTInteger64
- OFTReal
- OFTString
- OFTDate
- OFTTime
- OFTDateTime

5.59.9 Paging

Features can be 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.

5.59.10 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.

5.59.11 Open options

The following open options are available:

- USERPWD - Username and password, separated by colon.
- PAGE_SIZE=-1 - Limit feature count while fetching from server. Default value is -1 - no limit.
- BATCH_SIZE=-1 - Size of feature insert and update operations cache before send to server. If batch size is -1 batch mode is disabled. Default value is -1.
- NATIVE_DATA=NO - Whether to store the json *extensions* key in feature native data. Default value is NO.
- JSON_DEPTH=32 - The depth of json response that can be parsed. If depth is greater than this value, parse error occurs.

5.59.12 Dataset creation options

The following dataset/datasource creation options are available:

- KEY - Key value. Must be unique in whole NextGIS Web instance. Optional.
- DESCRIPTION - Resource description. Optional.
- USERPWD - Username and password, separated by colon.
- PAGE_SIZE=-1 - Limit feature count while fetching from server. Default value is -1 - no limit.
- BATCH_SIZE=-1 - Size of feature insert and update operations cache before send to server. If batch size is -1 batch mode is disabled. Default value is -1.

- NATIVE_DATA=NO - Whether to store the json *extensions* key in feature native data. Default value is NO.
- JSON_DEPTH=32 - The depth of json response that can be parsed. If depth is greater than this value, parse error occurs.

5.59.13 Layer creation options

The following layer creation options are available:

- OVERWRITE - Whether to overwrite an existing table with the layer name to be created. The resource will delete and new one will be created. This leads that resource identifier will change. Defaults to NO. Optional.
- KEY - Key value. Must be unique in whole NextGIS Web instance. Optional.
- DESCRIPTION - Resource description. Optional.

5.59.14 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 (alias, identifier, label field, grid visibility) map to layer metadata the following way:

- field alias -> FIELD_{field number}_ALIAS (for example FIELD_0_ALIAS)
- identifier -> FIELD_{field number}_ID (for example FIELD_0_ID)
- label field -> FIELD_{field number}_LABEL_FIELD (for example FIELD_0_LABEL_FIELD)
- grid visibility -> FIELD_{field number}_GRID_VISIBILITY (for example FIELD_0_GRID_VISIBILITY)

5.59.15 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:' prefix to where clause and 'fld_' prefix to field name.

Dataset supports ExecuteSQL method. Only the following queries are supported:

- DELAYER: layer_name; - delete layer with layer_name.
- DELETE FROM layer_name; - delete any features from layer with layer_name.
- 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 SetAttribute-Filter method input.

5.59.16 Examples

Read datasource contents (1730 is resource group identifier):

```
ogrinfo -ro NGW:https://demo.nextgis.com/resource/1730
```

Read layer details (1730 is resource group identifier, *Parks* is vector layer name):

```
ogrinfo -ro -so NGW:https://demo.nextgis.com/resource/1730 Parks
```

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.

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
```

5.59.17 See also

- *Raster side of the driver*
- NextGIS Web documentation
- NextGIS Web for developers

5.60 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.

5.60.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

5.60.2 See Also

- [General UK NTF Information](#)

5.60.3 Implementation Notes

5.60.3.1 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:

```
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
```

(continues on next page)

(continued from previous page)

```

    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.

5.60.3.2 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.

5.60.3.3 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:

```

(continues on next page)

(continued from previous page)

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.

5.60.3.4 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
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`”.

5.61 OGC API - Features

New in version 2.3.

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.

5.61.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

5.61.2 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}.

5.61.3 Layer schema

OGR needs a fixed schema per layer, but OGC API - Features Core doesn't impose fixed schema. So the driver will retrieve the first page of features (10 features) and establish a schema from this.

5.61.4 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.

5.61.5 Open options

The following options are available:

- **URL=**url: 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: Number of features to retrieve per request. Defaults to 10. Minimum is 1, maximum 10000.
- **USERPWD=**: 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.

5.61.6 Examples

- Listing the types of a OGC API - Features server :

```
$ ogrinfo OAPIF:https://www.ldproxy.nrw.de/rest/services/kataster

INFO: Open of `OAPIF:https://www.ldproxy.nrw.de/rest/services/kataster'
      using driver `OAPIF' successful.
1: flurstueck (Multi Polygon)
2: gebaeudebauwerk (Multi Polygon)
3: verwaltungseinheit (Multi Polygon)
```

- Listing the summary information of a OGC API - Features layer :

```
$ ogrinfo -al -so OAPIF:https://www.ldproxy.nrw.de/rest/services/kataster_
↳flurstueck

Layer name: flurstueck
Metadata:
  TITLE=Flurstück
Geometry: Multi Polygon
Feature Count: 9308456
Extent: (5.612726, 50.237351) - (9.589634, 52.528630)
Layer SRS WKT:
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"]]
id: String (0.0)
aktualit: Date (0.0)
flaeche: Real (0.0)
flstkennz: String (0.0)
land: String (0.0)
gemarkung: String (0.0)
flur: String (0.0)
flurstnr: String (0.0)
gmdschl: String (0.0)
```

(continues on next page)

(continued from previous page)

```

regbezirk: String (0.0)
kreis: String (0.0)
gemeinde: String (0.0)
lagebeztxt: String (0.0)
tntxt: String (0.0)

```

- 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://www.ldproxy.nrw.de/rest/services/kataster flurstueck -al -
↳q -where "flur = '028'"
Layer name: flurstueck
Metadata:
  TITLE=Flurstück
OGRFeature(flurstueck):1
  id (String) = DENW19AL0000geMFFL
  aktualit (Date) = 2017/04/26
  flaeche (Real) = 1739
  flstkennz (String) = 05297001600193_____
  land (String) = Nordrhein-Westfalen
  gemarkung (String) = Wünnenberg
  flur (String) = 016
  flurstnr (String) = 193
  gmdschl (String) = 05774040
  regbezirk (String) = Detmold
  kreis (String) = Paderborn
  gemeinde (String) = Bad Wünnenberg
  lagebeztxt (String) = Bleiwätscher Straße
  tntxt (String) = Platz / Parkplatz;1739
  MULTIPOLYGON (((8.71191 51.491084,8.7123 51.491067,8.712385 51.491645,8.712014,
↳51.491666,8.711993 51.491603,8.71196 51.491396,8.711953 51.491352,8.71191 51.
↳491084)))
[...]
```

- Spatial filtering

```

$ ogrinfo OAPIF:https://www.ldproxy.nrw.de/rest/services/kataster flurstueck -al -
↳q -spat 8.7 51.4 8.8 51.5
Layer name: flurstueck
Metadata:
  TITLE=Flurstück
OGRFeature(flurstueck):1
  id (String) = DENW19AL0000ht7LFL
  aktualit (Date) = 2013/02/19
  flaeche (Real) = 18
  flstkennz (String) = 05292602900206_____
  land (String) = Nordrhein-Westfalen
  gemarkung (String) = Fürstenberg
  flur (String) = 029
  flurstnr (String) = 206
  gmdschl (String) = 05774040
  regbezirk (String) = Detmold
  kreis (String) = Paderborn
  gemeinde (String) = Bad Wünnenberg

```

(continues on next page)

(continued from previous page)

```

lagebeztxt (String) = Karpke
tntxt (String) = Fließgewässer / Bach;18
MULTIPOLYGON (((8.768521 51.494915,8.768535 51.494882,8.768569 51.494908,8.
↪768563 51.494925,8.768521 51.494915)))
[...]
```

5.61.7 See Also

- “OGC API - Features - Part 1: Core” Standard
- *WFS (1.0,1.1,2.0) driver documentation*

5.62 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.

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 table are not accessible unless explicitly listed in the datasource name. Even in databases where all desired layers are in the ALL_SDO_GEOM_METADATA table, 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.

5.62.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

5.62.2 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.

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.

5.62.3 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 variable (i.e. 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)

5.62.4 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.

5.62.4.1 Layer Creation Options

- **OVERWRITE:** This may be “YES” to force an existing layer (=table) of the same desired name to be destroyed before creating the requested layer. The default value is “NO”
- **TRUNCATE:** 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. The default value is “NO”.
- **LAUNDER:** 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 “_”. The default value is “NO”.
- **PRECISION:** 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. The default is “YES”.
- **DIM:** This may be set to 2 or 3 to force the dimension of the created layer. Prior to GDAL 2.2, 3 is used by default. Starting with GDAL 2.2, the dimension of the layer geometry type is used by default.
- **SPATIAL_INDEX:** This may be set to FALSE 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”. Note: option was called INDEX in releases before GDAL 2
- **INDEX_PARAMETERS:** 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:** (starting with GDAL 2.0) 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). Defaults to YES.
- **DIMINFO_X:** 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:** 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:** 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:** 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 if it does not exactly match the WKT the driver expects.
- **MULTI_LOAD:** 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:** 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**: Define the first numeric value of the id column on the first rows. It's also work as a open option when used to append or update an existing dataset.
- **NO_LOGGING**: Define that the table and the geometry will be create with nologging attributes.
- **LOADER_FILE**: 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**: 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.

5.62.4.2 Layer Open Options

- **FIRST_ID**: See Layer Create Options comments on FIRST_ID.
- **MULTI_LOAD**: See Layer Create Options comments on MULTI_LOAD.
- **MULTI_LOAD_COUNT**: See Layer Create Options comments on MULTI_LOAD_COUNT.
- **WORKSPACE**: Define what user workspace to use.

5.62.4.3 Example

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
```

This second example loads a political boundaries layer from VPF (via the *OGDI driver*), and renames the layer from the cryptic OGDI layer name to something more sensible. If an existing table of the desired name exists it is overwritten.

```
% ogr2ogr -f OCI OCI:warmerda/password \  
  gltp:/vrf/usr4/mpp1/v0eur/vmaplv0/eurnasia \  
  -lco OVERWRITE=yes -nln polbndl_bnd 'polbndl@bnd(*)_line'
```

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'"
```

5.62.4.4 Credits

I would like to thank [SRC, LLC](#) for its financial support of the development of this driver.

5.63 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(geometrycolname),...:srs_tablename(sridcolumn,srtextcolumn)**. 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.

5.63.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

5.63.2 Access Databases (.MDB) support

Starting with GDAL 1.10, and on Windows provided that the “Microsoft Access Driver (*.mdb)” ODBC driver is installed, non-spatial MS Access Databases (not Personal Geodatabases or Geomedia databases) can be opened directly by their filenames.

5.63.3 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.

5.63.3.1 See Also

- [MSDN ODBC API Reference](#)
- *PGeo driver*
- *Geomedia driver*
- *MDB driver*

5.64 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.

5.64.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

5.64.2 Configuration options

- `OGR_ODS_HEADERS = FORCE / DISABLE / 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_ODS_FIELD_TYPES = STRING / 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.

5.65 OGD I Vectors

Driver short name

OGDI

Build dependencies

OGDI library

OGDI vector support is optional in OGR, and is normally only configured if OGD I is installed on the build system. If available OGD I vectors are supported for read access for the following family types:

- Point
- Line
- Area
- Text (Currently returned as points with the text in the “text” attribute)

OGDI can (among other formats) read VPF products, such as DCW and VMAP.

If an OGD I gltp url is opened directly the OGD I 3.1 capabilities for the driver/server are queried to get a list of layers. One OGR layer is created for each OGD I family available for each layer in the datastore. For drivers such as VRF this can result in a lot of layers. Each of the layers has an OGR name based on the OGD I name plus an underscore and the family name. For instance a layer might be called **watrcrsl@hydro(*)_line** if coming out of the VRF driver.

From GDAL/OGR 1.8.0, setting the `OGR_OGD I_LAUNDER_LAYER_NAMES` configuration option (or environment variable) to YES causes the layer names to be simplified. For example : `watrcrsl_hydro` instead of ‘watrcrsl@hydro(*)_line’

Alternatively to accessing all the layers in a datastore, it is possible to open a particular layer using a customized filename consisting of the regular GLTP URL to which you append the layer name and family type (separated by colons). This mechanism must be used to access layers of pre OGD 3.1 drivers as before OGD 3.1 there was no regular way to discover available layers in OGD.

```
gltp://<hostname>/<driver_name>/<dataset_name>:<layer_name>:<family>
```

Where <layer_name> is the OGD Layer name, and <family> is one of: “line”, “area”, “point”, or “text”.

OGD coordinate system information is supported for most coordinate systems. A warning will be produced when a layer is opened if the coordinate system cannot be translated.

There is no update or creation support in the OGD driver.

5.65.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

5.65.2 Error handling

Starting with GDAL 2.2 and OGD > 3.2.0beta2, if the OGD_STOP_ON_ERROR environment variable is set to NO, some errors can be gracefully recovered by OGD (in VPF driver). They will still be caught by GDAL and emitted as regular GDAL errors.

Note: be aware that this is a work in progress. Not all recoverable errors can be recovered, and some errors might be recovered silently.

5.65.3 Examples

Usage example ‘ogrinfo’:

```
ogrinfo gltp:/vrf/usr4/mpp1/v0eur/vmaplv0/eurnasia 'watrcrsl@hydro(*)_line'
```

In the dataset name ‘gltp:/vrf/usr4/mpp1/v0eur/vmaplv0/eurnasia’ the gltp:/vrf part is not really in the filesystem, but has to be added. The VPF data was at /usr4/mpp1/v0eur/. The ‘eurnasia’ directory should be at the same level as the dht. and lat. files. The ‘hydro’ reference is a subdirectory of ‘eurnasia/’ where watrcrsl.* is found.

Usage examples VMAP0 to SHAPE conversion with ‘ogr2ogr’:

```
ogr2ogr watrcrsl.shp gltp:/vrf/usr4/mpp1/v0eur/vmaplv0/eurnasia 'watrcrsl@hydro(*)_
↳line'
ogr2ogr polbnda.shp gltp:/vrf/usr4/mpp1/v0eur/vmaplv0/eurnasia 'polbnda@bnd(*)_area'
```

An OGR SQL query against a VMAP dataset. Again, note the careful quoting of the layer name.

```
ogrinfo -ro gltp:/vrf/usr4/mpp1/v0noa/vmaplv0/noamer \
-sql 'select * from "polbndl@bnd(*)_line" where use=26'
```

5.65.4 See Also

- [OGDI.SourceForge.Net](#)
- [VMap0 Coverages](#)

5.66 OpenAir - OpenAir Special Use Airspace Format

Driver short name

OpenAir

Driver built-in by default

This driver is built-in by default

This driver reads files describing Special Use Airspaces in the OpenAir format

Airspace are returned as features of a single layer called 'airspaces', with a geometry of type Polygon and the following fields: CLASS, NAME, FLOOR, CEILING.

Airspace geometries made of arcs will be tessellated. Styling information when present is returned at the feature level.

An extra layer called 'labels' will contain a feature for each label (AT element). There can be multiple AT records for a single airspace segment. The fields are the same as the 'airspaces' layer.

5.66.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

5.66.2 See Also

- [Description of OpenAir format](#)

5.67 ESRI File Geodatabase (OpenFileGDB)

Driver short name

OpenFileGDB

Driver built-in by default

This driver is built-in by default

The OpenFileGDB driver provides read access to vector layers of File Geodatabases (.gdb directories) created by ArcGIS 9 and above. 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 with GDAL \geq 2.2.

5.67.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

5.67.2 Spatial filtering

The driver cannot use the .spx files (when they are present) for spatial filtering. However, it will use the minimum bounding rectangle included at the beginning of the geometry blobs to speed up spatial filtering. By default, it will also build 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.

5.67.3 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.

5.67.3.1 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)

5.67.4 Comparison with the FileGDB driver

(Comparison done with a FileGDB driver using FileGDB API SDK 1.4)

Advantages of the OpenFileGDB 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.

Drawbacks of the OpenFileGDB driver:

- Read-only.
- Cannot use spatial indexes.
- Cannot read data from compressed data in CDF format (Compressed Data Format).

5.67.5 Examples

- Read 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"
```

- Get detailed info for FileGDB:

```
ogrinfo -al "C:\somefolder\MyGDB.gdb"
```

- Get detailed info for a zipped FileGDB:

```
ogrinfo -al "C:\somefolder\MyGDB.gdb.zip"
```

5.67.6 Links

- *FileGDB driver*, relying on the FileGDB API SDK
- Reverse-engineered specification of the [FileGDB format](#)

5.68 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 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.

5.68.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

5.68.2 Configuration

In the *data* folder of the GDAL distribution, you can find a *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.

Starting with GDAL 2.0, fields can be computed with SQL expressions (evaluated by SQLite engine) from other fields/tags. For example to compute the *z_order* attribute.

5.68.2.1 “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*.

For example :

```
ogr2ogr -f PostgreSQL "PG:dbname=osm" test.pbf -lco COLUMN_TYPES=other_tags=hstore
```

5.68.2.2 “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”.

5.68.3 Internal working and performance tweaking

The driver will use an internal SQLite database to resolve geometries. If that database remains under 100 MB 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. The 100 MB default threshold can be adjusted with the OSM_MAX_TMPFILE_SIZE configuration option (value in MB).

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 defining the OSM_USE_CUSTOM_INDEXING configuration option to NO.

When custom indexing is used (default case), the OSM_COMPRESS_NODES configuration option can be set to YES (the default is NO). This option might be turned on to improve performances 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.

5.68.4 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.

Starting with GDAL 2.2, 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.

5.68.5 Spatial filtering

Due to way `.osm` or `.pbk` 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.

5.68.6 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 `.pbk` file without downloading it :

```
wget -O - http://www.example.com/some.pbf | ogr2ogr -f SQLite my.sqlite /vsistdin/
```

or

(continues on next page)

(continued from previous page)

```
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 | bzipcat | ogr2ogr -f SQLite my.sqlite /
↳vsistdin/
```

5.68.7 Open options

- **CONFIG_FILE=filename:** (GDAL >=2.0) Configuration filename. Defaults to {GDAL_DATA}/osmconf.ini.
- **USE_CUSTOM_INDEXING=YES/NO:** (GDAL >=2.0) Whether to enable custom indexing. Defaults to YES.
- **COMPRESS_NODES=YES/NO:** (GDAL >=2.0) Whether to compress nodes in temporary DB. Defaults to NO.
- **MAX_TMPFILE_SIZE=int_val:** (GDAL >=2.0) Maximum size in MB of in-memory temporary file. If it exceeds that value, it will go to disk. Defaults to 100.
- **INTERLEAVED_READING=YES/NO:** (GDAL >=2.0) Whether to enable interleaved reading. Defaults to NO.

5.68.8 See Also

- [OpenStreetMap home page](#)
- [OSM XML Format description](#)
- [OSM PBF Format description](#)

5.69 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.

5.69.1 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.*)

5.69.2 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.

5.69.3 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	GDAL >= 2.3.0: 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) GDAL <= 2.2.x: text (t), limited to ASCII strings; font size (s); text color (c); x and y offsets (dx, dy); angle (a)	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.

5.69.4 See Also

- [PDF raster](#) documentation page
- [Feature Style Specification](#)

5.70 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.

5.70.1 Driver capabilities

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

5.70.2 See Also

- [Description of PDS format](#) (see Annex A.29 from StdRef_20090227_v3.8.pdf)

5.71 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.

Starting with OGR 1.11, the PGDump driver supports creating tables with multiple PostGIS geometry columns (following [rfc-41](#))

5.71.1 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.*)

5.71.2 Creation options

5.71.2.1 Dataset Creation Options

- **LINEFORMAT**: 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 layer creation option which may have a value of **CRLF** (DOS format) or **LF** (Unix format).

5.71.2.2 Layer Creation Options

- **GEOM_TYPE**: 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. “geometry” is the default value.
- **LAUNDER**: 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. The default value is “YES”. If enabled the table (layer) name will also be laundered.
- **PRECISION**: 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**: 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**: 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**: (OGR >= 1.8.1) 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** (starting with GDAL 2.4) or YES/NO for earlier versions and backward compatibility: Set to GIST (GDAL >= 2.4, or YES for earlier versions) 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 (GDAL >= 2.4, or FALSE for earlier versions) 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**: Set to OFF by default. Creates a temporary table instead of a permanent one.
- **UNLOGGED**: (From GDAL 2.0) 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.
- **WRITE_EWKT_GEOM**: Set to OFF by default. Turn to ON to write EWKT geometries instead of HEX geometries. This option will have no effect if PG_USE_COPY environment variable is to YES.

- **CREATE_TABLE**: 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**: (OGR >= 1.8.1) 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 (default in GDAL 2.0) in order DROP TABLE IF EXISTS to be emitted (needs PostgreSQL >= 8.2)
- **SRID**: 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**: (From GDAL 1.9.0) Can be set to TRUE to force non-spatial layers (wkbNone) to be created as spatial tables of type GEOMETRY (wkbUnknown), which was the behaviour prior to GDAL 1.8.0. Defaults to NO, in which case a regular table is created and not recorded in the PostGIS geometry_columns table.
- **FID**: (From GDAL 1.9.0) Name of the FID column to create. Defaults to 'ogc_fid'.
- **FID64**: (From GDAL 2.0) This may be "TRUE" to create a FID column that can support 64 bit identifiers. The default value is "FALSE".
- **EXTRACT_SCHEMA_FROM_LAYER_NAME**: (From GDAL 1.9.0) Can be set to NO to avoid considering the dot character as the separator between the schema and the table name. Defaults to YES.
- **COLUMN_TYPES**: (From GDAL 1.10) 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**: (From GDAL 1.9.0) Can be set to 2.0 for PostGIS 2.0 compatibility. Starting with GDAL 2.0, it is important to set it correctly when dealing with non-linear geometry types. Starting with GDAL 2.1, can be POSTGIS_VERSION=2.2 is specially dealt with to correctly export POINT EMPTY geometries
- **DESCRIPTION** (From GDAL 2.1) 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.

5.71.2.3 Environment variables

- **PG_USE_COPY**: This may be "YES" for using COPY for inserting data to Postgresql. COPY is significantly faster than INSERT.

5.71.2.4 VSI Virtual File System API support

(Some features below might require OGR >= 1.9.0)

The driver supports rwriting 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.

5.71.2.5 Example

- 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
```

- Pipe 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 -
```

5.71.2.6 See Also

- [OGR PostgreSQL driver Page](#)
- [PostgreSQL Home Page](#)
- [PostGIS](#)
- [PostGIS / OGR Wiki Examples Page](#)

5.72 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 drivers 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. On Windows, no ODBC DSN is required. On Linux, there are problems with DSN-less connection due to incomplete or buggy implementation of this feature in the [MDB Tools](#) package. So, it is required to configure Data Source Name (DSN) if the MDB Tools driver is used (check instructions below).

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 programmatically 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. Measures information will be discarded. 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.

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.

5.72.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

5.72.2 How to use PGeo driver with unixODBC and MDB Tools (on Unix and Linux)

Starting with GDAL/OGR 1.9.0, the *MDB* driver is an alternate way of reading ESRI Personal GeoDatabase .mdb files without requiring unixODBC and MDB Tools

This article gives step-by-step explanation of how to use OGR with unixODBC package and how to access Personal Geodatabase with PGeo driver. See also [GDAL wiki for other details](#)

5.72.2.1 Prerequisites

1. Install [unixODBC](#) >= 2.2.11
2. Install [MDB Tools](#) >= 0.6. I also tested with 0.5.99 (0.6 pre-release).

(On Ubuntu 8.04 : `sudo apt-get install unixodbc libmdbodbc`)

5.72.2.2 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.

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
CPOutput    =
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

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.

5.72.2.3 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. As a data source use PGeo:<DSN> where <DSN> is a name of DSN entry you put to your .odbc.ini.

```
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: ...
```

5.72.3 Resources

- [About ESRI Geodatabase](#)
- [\[mdbtools-dev\] DSN-less connection not supported?](#)

5.72.4 See also

- [MDB driver page](#)

5.73 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.

5.73.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports Georeferencing

This driver supports georeferencing

5.73.2 Connecting to a database

To connect to a Postgres datasource, use a connection string specifying the database name, with additional parameters as necessary

```
PG:dbname=databasename
```

or

```
PG:"dbname='databasename' host='addr' port='5432' user='x' password='y'"
```

or starting with GDAL 3.1:

```
PG:service=servicename
```

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, so any other options and defaults that would apply to it, apply to the name here (refer to the documentation of the PostgreSQL server. Here for [PostgreSQL 8.4](#)). The PG: prefix is used to mark the name as a postgres connection string.

5.73.3 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.

Starting with GDAL 1.7.0, the driver also supports the [geography](#) column type introduced in PostGIS 1.5.

Starting with GDAL 2.0, the driver also supports reading and writing the following non-linear geometry types :CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE and MULTISURFACE

5.73.4 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.

The PostgreSQL driver in OGR supports the `OGRDataSource::StartTransaction()`, `OGRDataSource::CommitTransaction()` and `OGRDataSource::RollbackTransaction()` calls in the normal SQL sense.

5.73.5 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.

5.73.5.1 Dataset open options

(GDAL >= 2.0)

- **DBNAME**=string: Database name.
- **PORT**=integer: Port.
- **USER**=string: User name.
- **PASSWORD**=string: Password.
- **HOST**=string: Server hostname.
- **DBNAME**=string: Database name.
- **SERVICE**=string: Service name (GDAL >= 3.1)
- **ACTIVE_SCHEMA**=string: Active schema.
- **SCHEMAS**=string: Restricted sets of schemas to explore (comma separated).
- **TABLES**=string: Restricted set of tables to list (comma separated).
- **LIST_ALL_TABLES**=YES/NO: This may be “YES” to force all tables, including non-spatial ones, to be listed.
- **PRELUDE_STATEMENTS**=string (GDAL >= 2.1). SQL statement(s) to send on the PostgreSQL client connection before any other ones. In case of several statement, 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**=string (GDAL >= 2.1). SQL statement(s) to send on the PostgreSQL client connection after any other ones. In case of several statement, they must be separated with the semi-column (;) sign. With the above example value for PRELUDE_STATEMENTS, the appropriate CLOSING_STATEMENTS would be "COMMIT".

5.73.5.2 Dataset Creation Options

None

5.73.5.3 Layer Creation Options

- **GEOM_TYPE**: The GEOM_TYPE layer creation option can be set to one of "geometry", "geography" (PostGIS >= 1.5), "BYTEA" or "OID" to force the type of geometry used for a table. For a PostGIS database, "geometry" is the default value.
- **OVERWRITE**: This may be "YES" to force an existing layer of the desired name to be destroyed before creating the requested layer.
- **LAUNDER**: 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. The default value is "YES". If enabled the table (layer) name will also be laundered.
- **PRECISION**: 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**: 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**: 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 (starting with GDAL 2.4) or YES/NO for earlier versions and backward compatibility: Set to GIST (GDAL >=2.4, or YES for earlier versions) 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 (GDAL >= 2.4, or FALSE for earlier versions) 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**: (From GDAL 1.8.0) Set to OFF by default. Creates a temporary table instead of a permanent one.
- **UNLOGGED**: (From GDAL 2.0) 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**: (From GDAL 1.8.1) Can be set to TRUE to force non-spatial layers (wkbNone) to be created as spatial tables of type GEOMETRY (wkbUnknown), which was the behaviour prior to GDAL 1.8.0. Defaults to NO, in which case a regular table is created and not recorded in the PostGIS geometry_columns table.

- **FID:** (From GDAL 1.9.0) Name of the FID column to create. Defaults to 'ogc_fid'.
- **FID64:** (From GDAL 2.0) This may be "TRUE" to create a FID column that can support 64 bit identifiers. The default value is "FALSE".
- **EXTRACT_SCHEMA_FROM_LAYER_NAME:** (From GDAL 1.9.0) Can be set to NO to avoid considering the dot character as the separator between the schema and the table name. Defaults to YES.
- **COLUMN_TYPES:** (From GDAL 1.10) 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** (From GDAL 2.1) 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.

5.73.5.4 Configuration Options

There are a variety of [Configuration Options](#) which help control the behavior of this driver.

- **PG_USE_COPY:** This may be "YES" for using COPY for inserting data to PostgreSQL. COPY is significantly faster than INSERT. Starting with GDAL 2.0, COPY is used by default when inserting from a table that has just been created.
- **PGSQL_OGR_FID:** 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_USE_BASE64:** (GDAL >= 1.8.0) 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_TRUNCATE:** (GDAL >= 1.11) 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".

5.73.5.5 Examples

- 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
```

- This second example loads a political boundaries layer from VPF (via the [OGDI driver](#)), and renames the layer from the cryptic OGD layer name to something more sensible. If an existing table of the desired name exists it is overwritten.

```
% ogr2ogr -f PostgreSQL PG:dbname=warmerda \
    gltp:/vrf/usr4/mpp1/v0eur/vmaplv0/eurnasia \
    -lco OVERWRITE=yes -nln polbndl_bnd 'polbndl@bnd(*)_line'
```

- Export a single Postgres table to GeoPackage:

```
ogr2ogr \
  -f GPKG output.gpkg \
  PG:dbname="my_database" "my_table"
```

- Export many Postgres tables to GeoPackage:

```
ogr2ogr \
  -f GPKG output.gpkg \
  PG:'dbname=my_database tables=table_1,table_3'
```

- Export a whole Postgres database to GeoPackage:

```
ogr2ogr \
  -f GPKG output.gpkg \
  PG:dbname=my_database
```

- Load a single layer GeoPackage into Postgres:

```
ogr2ogr \
  -f "PostgreSQL" PG:dbname="my_database" \
  input.gpkg \
  -nln "name_of_new_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
```

- 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'"
```

- 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'
```

- This example shows use of PRELUDE_STATEMENTS and CLOSING_STATEMENTS as destination open options of ogr2ogr.

```
ogrinfo "pg:dbname=mydb" poly.shp -doo "PRELUDE_STATEMENTS=BEGIN; SET LOCAL_
↪statement_timeout TO '1h';" -doo CLOSING_STATEMENTS=COMMIT
```

5.73.6 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`)

5.73.7 See Also

- [Advanced OGR PostgreSQL driver Information](#)
- [OGR PostgreSQL SQL Dump driver Page](#)
- [PostgreSQL Home Page](#)
- [PostGIS](#)
- [PostGIS / OGR Wiki Examples Page](#)

5.73.7.1 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

Starting with GDAL 1.8.0, 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_column_name2)],...]` 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 behaviour is that you must be careful if you re-use 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())
```

OGR < 1.8.0 behaviour can be obtained by setting the configuration option `OGR_PG_RETRIEVE_FID` to `FALSE`.

Issues with transactions

Note: this section mostly applies to GDAL 2.0, that implements rfc-54 Previous versions had different behaviour which made it impractical to handle both reading and writing with the same OGR datasource. Reading several layers in a interleaved way was also not working properly. The new below behaviour should enable more powerful uses, but might cause subtle problems for existing code that relied on implicit transactions being regularly flushed by the PG driver in GDAL 1.X

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 behaviour 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

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

```

(continues on next page)

(continued from previous page)

```

lyr2->ResetReading()          CLOSE cur2

ds->CommitTransaction()       COMMIT

~~~~~

ds->StartTransaction()        BEGIN

lyr1->GetNextFeature()         DECLARE cur1 CURSOR FOR SELECT * FROM lyr1
                              FETCH 1 IN cur1

lyr1->CreateFeature(f)         INSERT INTO cur1 ...

ds->CommitTransaction()       CLOSE cur1 (implicit)
                              COMMIT

lyr1->GetNextFeature()         FETCH 1 IN cur1      ==> Error since the cursor_
↳ was closed with the commit. Explicit ResetReading() required before

~~~~~

lyr1->GetNextFeature()         BEGIN (implicit)
                              DECLARE cur1 CURSOR FOR SELECT * FROM lyr1
                              FETCH 1 IN cur1

ds->StartTransaction()        SAVEPOINT savepoint

lyr1->CreateFeature(f)         INSERT INTO cur1 ...

ds->CommitTransaction()       RELEASE SAVEPOINT savepoint

lyr1->ResetReading()          CLOSE cur1
                              COMMIT (implicit)

```

Note: in reality, the PG drivers fetches 500 features at once. The FETCH 1 is for clarity of the explanation.

Advanced Examples

- This example shows using ogrinfo to list only the layers specified by the *tables=* options.

```
ogrinfo -ro PG:'dbname=warmerda tables=table1,table2'
```

- This example shows using ogrinfo to query a table 'foo' with multiple geometry columns ('geom1' and 'geom2').

```
ogrinfo -ro -al PG:dbname=warmerda 'foo(geom2)'
```

- This example show how to list only the layers inside the schema apt200810 and apt200812. The layer names will be prefixed by the name of the schema they belong to.

```
ogrinfo -ro PG:'dbname=warmerda schemas=apt200810,apt200812'
```

- This example shows using ogrinfo to list only the layers inside the schema named apt200810. Note that the layer names will not be prefixed by apt200810 as only one schema is listed.

```
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 -nlm to specify the qualified layer name.

```
ogr2ogr -overwrite -f PostgreSQL "PG:dbname=warmerda" mytable.shp mytable -nlm ↵  
↪myschema.mytable
```

Note that using -lco SCHEMA=mytable instead of -nlm 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 -nlm 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*

5.74 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:

5.74.1 PLScenes (Planet Labs Scenes), Data V1 API

New in version 2.2.

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.

5.74.1.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

5.74.1.2 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.
- **metadata**=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.

5.74.1.3 Open options

The following open options are available :

- **VERSION**=data_v1: To specify the API version to request.
- **API_KEY**=value: To specify the Planet 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.
- **RANDOM_ACCESS=YES/NO**: Whether raster should be accessed in random access mode (but with potentially not optimal throughput). If NO, in-memory ingestion is done. Default is YES.
- **ACTIVATION_TIMEOUT=int**: Number of seconds during which to wait for asset activation (raster). Default is 3600.
- **METADATA=YES/NO**: (Raster only) Whether scene metadata should be fetched from the API and attached to the raster dataset. Defaults to YES.

5.74.1.4 Configuration options

The following configuration options are available :

- **PL_API_KEY=value**: To specify the Planet API KEY.

5.74.1.5 Attributes

The layer field definition is built from the “plscensconf.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

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
```

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
```

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
```

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"
```

List all downloadable scenes:

```
ogrinfo -ro -al -q "PLScenes:" -oo API_KEY=some_value PSOrthoTile -where "permissions=
↳ 'assets:download' "
```

5.74.1.6 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

Displaying raster metadata :

```
gdalinfo "PLScenes:scene=scene_id,itemtypes=itemtypes,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
```

Converting/downloading a whole file:

```
gdal_translate "PLScenes:" -oo API_KEY=some_value -oo SCENE=scene_id \
               -oo ITEMYPES=itemtypes -oo ASSET=analytic -oo RANDOM_ACCESS=NO out.
↳tif
```

5.74.1.7 See Also

- *General documentation page for PLScenes driver*
- *Documentation of Planet Scenes Data API v1*
- *Raster PLMosaic / Planet Mosaics API driver*

5.74.2 See Also

- *Documentation of Planet Scenes Data API V1*
- *Raster PLMosaic / Planet Mosaics API driver*

5.75 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 run-time 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.

5.75.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (vsimem/, etc.)*

5.75.2 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, NIN-FOM, 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 indicates the object class of the feature.	All features
RVER	Record version.	
AGEN	Numeric agency code, such as 50 for the Canadian Hydrographic Service. A potentially outdated list is available in agencode.txt.	All features
FIDN	Feature identification number.	All features
FIDS	Feature identification subdivision.	All features
DSNM	Dataset name. The name of the file the feature came from. Used with LNAM to form a unique dataset wide identifier for a feature.	All features
INFORM	Informational text.	Some features

(continues on next page)

(continued from previous page)

NINFOM	Informational text in national language.	Some features
OBJNAM	Object name	Some features
NOBJNM	Object name in national language.	Some features
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, FIDN and FIDS used to uniquely identify this features within an S-57 file.	All features
LNAM_REFS	List of long names of related features	All Features
FFPT_RIND	Relationship indicators for each of the LNAM_REFS relationships.	All Features

5.75.3 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.

5.75.4 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.

Starting with GDAL 2.0, they can also be specified independently as open options to the driver.

- **UPDATES=APPLY/IGNORE:** Should update files be incorporated into the base data on the fly. Default is APPLY.
- **SPLIT_MULTIPPOINT=ON/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. Default is OFF.
- **ADD_SOUNDG_DEPTH=ON/OFF:** Should a DEPTH attribute be added on SOUNDG features and assign the depth of the sounding. This should only be enabled with SPLIT_MULTIPPOINT is also enabled. Default is OFF.
- **RETURN_PRIMITIVES=ON/OFF:** Should all the low level geometry primitives be returned as special IsolatedNode, ConnectedNode, Edge and Face layers. Default is OFF.

- **PRESERVE_EMPTY_NUMBERS=ON/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. Default is OFF.
- **LNAM_REFS=ON/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. Default is ON.
- **RETURN_LINKAGES=ON/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. Default is OFF.
- **RECODE_BY_DSSI=ON/OFF**: (OGR >= 1.10) Should attribute values be recoded to UTF-8 from the character encoding specified in the S57 DSSI record. Default is OFF.

Example:

```
set OGR_S57_OPTIONS = "RETURN_PRIMITIVES=ON, RETURN_LINKAGES=ON, LNAM_REFS=ON"
```

5.75.5 S-57 Export

Preliminary S-57 export capability has been added in GDAL/OGR 1.2.0 but 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"
```

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**: Exchange purpose. Default is 1.
- **S57_INTU**: Intended usage. Default is 4.
- **S57_EDTN**: Edition number. Default is 2.
- **S57_UPDN**: Update number. Default is 0.
- **S57_UADT**: Update application date. Default is 20030801.
- **S57_ISDT**: Issue date. Default is 20030801.
- **S57_STED**: Edition number of S-57. Default is 03.1.
- **S57_AGEN**: Producing agency. Default is 540.
- **S57_COMT**: Comment.
- **S57_AALL**: Lexical level used for the ATTF fields. Default is 0. (GDAL >= 2.4)
- **S57_NALL**: Lexical level used for the NATF fields. Default is 0. (GDAL >= 2.4)
- **S57_NOMR**: Number of meta records (objects with acronym starting with “M_”). Default is 0.
- **S57_NOGR**: Number of geo records. Default is 0.
- **S57_NOLR**: Number of collection records. Default is 0.
- **S57_NOIN**: Number of isolated node records. Default is 0.
- **S57_NOCN**: Number of connected node records. Default is 0.
- **S57_NOED**: Number of edge records. Default is 0.
- **S57_HDAT**: Horizontal geodetic datum. Default is 2.

- **S57_VDAT**: Vertical datum. Default is 17.
- **S57_SDAT**: Sounding datum. Default is 23.
- **S57_CSCL**: Compilation scale of data (1:X). Default is 52000.
- **S57_COMF**: Floating-point to integer multiplication factor for coordinate values. Default is 10000000. (GDAL >= 2.4)
- **S57_SOMF**: Floating point to integer multiplication factor for 3-D (sounding) values. Default is 10. (GDAL >= 2.4)

5.75.6 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 \(main\)](#)

5.76 ESRI ArcSDE

Driver short name

SDE

Build dependencies

ESRI SDE

OGR optionally supports reading ESRI ArcSDE database instances. ArcSDE is a middleware spatial solution for storing spatial data in a variety of backend relational databases. The OGR ArcSDE driver depends on being built with the ESRI provided ArcSDE client libraries.

ArcSDE instances are accessed with a datasource name of the following form. The server, instance, username and password fields are required. The instance is the port number of the SDE server, which generally defaults to 5151. If the layer parameter is specified then the SDE driver is able to skip reading the summary metadata for each layer; skipping this step can be a significant time savings.

Note: Only GDAL 1.6+ supports querying against versions and write operations. Older versions only support querying against the base (SDE.DEFAULT) version and no writing operations.

```
SDE:server,instance,database,username,password[,layer]
```

To specify a version to query against, you *must* specify a layer as well. The SDE.DEFAULT version will be used when no version name is specified.

```
SDE:server,instance,database,username,password,layer,[version]
```

You can also request to create a new version if it does not already exist. If the child version already exists, it will be used unless the SDE_VERSIONOVERWRITE environment variable is set to “TRUE”. In that case, the version will be deleted and recreated.

```
SDE:server,instance,database,username,password,layer,[parentversion],[childversion]
```

The OGR ArcSDE driver does not support reading CAD data (treated as BLOB attribute), annotation properties, measure values at vertices, or raster data. The `ExecuteSQL()` method does **not** get passed through to the underlying database. For now it is interpreted by the limited OGR SQL handler. Spatial indexes are used to accelerate spatial queries.

The driver has been tested with ArcSDE 9.x, and should work with newer versions, as well as ArcSDE 8.2 or 8.3. Both 2D and 3D geometries are supported. Curve geometries are approximated as line strings (actually still TODO).

ArcSDE is generally sensitive to case-specific, fully-qualified tablename. While you may be able to use short names for some operations, others (notably deleting) will require a fully-qualified name. Because of this fact, it is generally best to **always** use fully-qualified table names.

5.76.1 Layer Creation Options

- **OVERWRITE**: This can be set to allow an existing layer to be overwritten during the layer creation process. If set, and the value is not “NO”, the layer will first be deleted prior to creating a new layer of the same name as an existing layer. Set to “NO” explicitly, or do not include the option to treat attempts to create new layers which collide with existing layers of the same name as an error. Off by default.
- **GEOMETRY_NAME**: By default OGR creates new layers with the geometry (feature) column named ‘SHAPE’. If you wish to use a different name, it can be supplied with the `GEOMETRY_NAME` layer creation option.
- **SDE_FID**: Can be set to override the default name of the feature ID column. The default is “OBJECTID”.
- **SDE_KEYWORD**: The DBTUNE keyword with which to create the layer. Defaults to “DEFAULTS”.
- **SDE_DESCRIPTION**: The text description of the layer. Defaults to “Created by GDAL/OGR 1.6” (Also used as the version description when creating a new child version from a parent version.)
- **SDE_MULTIVERSION**: If this creation option is set is set to “FALSE”, multi-versioning will be disabled for the layer at creation time. By default, multiversion tables are created when layers are created on an SDE datasource.
- **USE_NSTRING**: If this option is set to “TRUE” then string fields will be created as type NSTRING. This option was added for GDAL/OGR 1.9.0.

5.76.2 Environment variables

- **OGR_SDE_GETLAYERTYPE**: This may be “TRUE” to determine the geometry type from the database. Otherwise, the SDE driver will always return an Unknown geometry type.
- **OGR_SDE_SEARCHORDER**: This may be “ATTRIBUTE_FIRST” to tell ArcSDE to filter based on attributes *before* using a spatial filter or “SPATIAL_FIRST” to use the spatial filter. By default, it uses the spatial filter first.
- **SDE_VERSIONOVERWRITE**: If set to “TRUE”, the specified child version will be deleted before being recreated. Note that this action does nothing to reconcile any edits that existed on that version before doing so and essentially throws them away.
- **OGR_SDE_USE_NSTRING**: If this option is set to “TRUE” then string fields will be created as type NSTRING. This option was added for GDAL/OGR 1.9.0.
-

5.76.3 Examples

See the `ogr_sde.py` test script for some example connection strings and usage of the driver.

5.77 SDTS

Driver short name

SDTS

Driver built-in by default

This driver is built-in by default

SDTS TVP (Topological Vector Profile) and Point Profile datasets are supported for read access. Each primary attribute, node (point), line and polygon module is treated as a distinct layer.

To select an SDTS transfer, the name of the catalog file should be used. For instance `TR01CATD.DDF` where the first four characters are all that typically varies.

SDTS coordinate system information is properly supported for most coordinate systems defined in SDTS.

There is no update or creation support in the SDTS driver.

Note that in TVP datasets the polygon geometry is formed from the geometry in the line modules. Primary attribute module attributes should be properly attached to their related node, line or polygon features, but can be accessed separately as their own layers.

This driver has no support for raster (DEM) SDTS datasets.

5.77.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

5.77.2 See Also

- [SDTS Abstraction Library](#): The base library used to implement this driver.
- <http://mcmcweb.er.usgs.gov/sdts>: Main USGS SDTS web page.

5.78 SEG-P1 / UKOOA P1/90

Driver short name

SEGUKOOA

Driver built-in by default

This driver is built-in by default

This driver reads files in SEG-P1 and UKOOA P1/90 formats. Those files are simple ASCII files that contain seismic shotpoints. Two layers are reported : one with the points, and another ones where sequential points with same line name are merged together in a single feature with a line geometry.

5.78.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (vsimem/, etc.)*

5.78.2 See Also

- [Description of SEG-P1 format](#)
- [Description of UKOOA P1/90 format](#)

5.79 SEG-Y / SEG-Y

Driver short name

SEG-Y

Driver built-in by default

This driver is built-in by default

This driver reads files in SEG-Y format. Those files are binary files that contain single-line seismic digital data. The driver will report the attributes of the trace header (in their raw form, see the SEG-Y specification for more information), and use the receiver group coordinates as geometry. The sample values are also reported.

A layer “{basefilename}_header” is also created and contains a single feature with the content of the text and binary file headers.

5.79.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

5.79.2 See Also

- [Description of SEG-Y format Rev1](#)
- [Wikipedia page about SEG-Y format](#)

5.80 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.

5.80.1 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.*)

5.80.2 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.

5.80.3 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.

5.80.3.1 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 $2^{24}.a+2^{16}.b+2^8.c+d$.
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: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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

5.80.3.2 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.

5.80.4 Mapping between file and layers

5.80.4.1 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;
- ...

5.80.4.2 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.

5.80.5 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]	4 th , 11 th , and last two time steps, for both nodes and elements

5.80.6 Datasource creation options

Datasource creation options can be specified with the “-dsco” flag in ogr2ogr.

TITLE 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 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.`

5.80.7 Layer creation options

Layer creation options can be specified with the “-lco” flag in ogr2ogr.

DATE 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 datasource creation option is: `-lco DATE=24000.`

5.80.8 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 behaviour depends on the type of layer (points or elements). The following table explains the behaviour of the driver in the different cases.

Op- era- tion	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 attributes 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 1/1000 th 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 not allow for quick insertions or deletion of features and the file must be recreated for each operation.

5.80.9 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.

5.80.10 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.

5.81 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 behaviour 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. Starting with GDAL 2.3, 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=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.

5.81.1 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`.

5.81.2 Open options

The following open options are available.

- **ENCODING**=encoding_name: to 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: Set to YES (default is NO) 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. (Starting with GDAL 2.1) 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: (OGR >= 2.2) Default to YES in GDAL 2.2. Whether the shapefile should be automatically repacked when needed, at dataset closing or at FlushCache()/SyncToDisk() time.
- **DBF_EOF_CHAR**=YES/NO: (OGR >= 2.2) Default to YES in GDAL 2.2. 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.

5.81.3 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.

Starting with OGR 1.10, 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.

5.81.4 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. See the [FAQ](#) for a solution.

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. Starting with version 1.7, 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_l

- 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.

Starting with GDAL 2.0, 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.

Starting with GDAL 2.2, REPACK is also done automatically at file closing, or at FlushCache()/SyncToDisk() time, since shapefiles with holes can cause interoperability issues with other software.

5.81.5 Field sizes

Starting with GDAL/OGR 1.10, 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.

5.81.6 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.

Note: RECOMPUTE EXTENT ON is available in OGR >= 1.9.0.

5.81.7 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.

Starting with OGR 1.11, 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.

5.81.8 Dataset Creation Options

None

5.81.9 Layer Creation Options

- **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. The measure support was added in GDAL 2.1. MULTIPATCH files are supported since GDAL 2.2.
- **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:** (OGR >= 1.10.0) set the YES to resize fields to their optimal size. See above “Field sizes” section. Defaults to NO.
- **2GB_LIMIT=YES/NO:** (OGR >= 1.11) set the YES to enforce the 2GB file size for .SHP or .DBF files. Defaults to NO.
- **SPATIAL_INDEX=YES/NO:** (OGR >= 2.0) set the YES to create a spatial index (.qix). Defaults to NO.
- **DBF_DATE_LAST_UPDATE=YYYY-MM-DD:** (OGR >= 2.0) Modification date to write in DBF header with year-month-day format. If not specified, current date is used. Note: behaviour of past GDAL releases was to write 1995-07-26
- **AUTO_REPACK=YES/NO:** (OGR >= 2.2) Default to YES in GDAL 2.2. Whether the shapefile should be automatically repacked when needed, at dataset closing or at FlushCache()/SyncToDisk() time.
- **DBF_EOF_CHAR=YES/NO:** (OGR >= 2.2) Default to YES in GDAL 2.2. 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.

5.81.10 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/`, `/vsizip/`, `/vsicurl/` domains.

5.81.11 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.

5.81.12 Examples

- A merge of two shapefiles ‘file1.shp’ and ‘file2.shp’ into a new file ‘file_merged.shp’ is performed 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.

- Building a spatial index :

```
% ogrinfo file1.shp -sql "CREATE SPATIAL INDEX ON file1"
```

- Resizing columns of a DBF file to their optimal size (OGR >= 1.10.0) :

```
% ogrinfo file1.dbf -sql "RESIZE file1"
```

5.81.13 Advanced topics

(GDAL >= 2.0) The `SHAPE_REWIND_ON_WRITE` configuration option/environment variable 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).

(GDAL >= 2.1) The `SHAPE_RESTORE_SHX` configuration option/environment variable can be set to `YES` (default `NO`) to restore broken or absent `.shx` file from associated `.shp` file during opening.

5.81.14 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.)*

5.81.15 See Also

- [Shapelib Page](#)
- [User Notes on OGR Shapefile Driver](#)

5.82 Norwegian SOSI Standard

Driver short name

SOSI

Build dependencies

FYBA library

This driver requires the FYBA library.

5.82.1 Open options

Starting with GDAL 3.1, the following open options can be specified (typically with the -oo name=value parameters of ogrinfo or ogr2ogr):

- appendFieldsMap=(defaults is 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.’

5.82.1.1 Examples

- 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
```

- This example will convert a sosi file to a shape a file where only duplicates for BEITEBRUKERID and OPPHAV will be appended with a comma between.

```
ogr2ogr -t_srs EPSG:4258 test_poly.shp test_duplicate_fields.sos polygons -oo_↪appendFieldsMap="BEITEBRUKERID&OPPHAV"
```

- 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:,"
```

5.83 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**”

Starting with GDAL 2.2, 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. Starting with OGR 1.10, datetime field types are also handled.

Starting with GDAL 2.2, 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]*”.

5.83.1 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.*)

5.83.2 “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.

5.83.3 Tables with multiple geometry columns

Layers with multiple geometry columns can be created, modified or read, following new API described in [rfc-41](#)

5.83.4 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.

5.83.5 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 $\geq 2.3.1$). 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
```

5.83.6 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).

5.83.7 The SQLite SQL dialect

Starting with OGR 1.10, the SQLite SQL engine can be used to run SQL queries on any OGR datasource if using the *SQL SQLite dialect*.

5.83.8 The VirtualOGR SQLite extension

Starting with OGR 1.10, 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('libspatialite.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('libspatialite.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.

5.83.9 Creation Issues

The SQLite driver supports creating new SQLite database files, or adding tables to existing ones.

5.83.9.1 Transaction support (GDAL >= 2.0)

The driver implements transactions at the database level, per rfc-54

5.83.9.2 Dataset open options

(GDAL >= 2.0)

- **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.

5.83.9.3 Database Creation Options

- **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:** (Starting with GDAL 1.7.0) 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: (Starting with GDAL 1.9.0) 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:** (Starting with GDAL 1.8.0) 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.

5.83.9.4 Layer Creation Options

- **FORMAT=WKB/WKT/SPATIALITE:** 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 starting with GDAL 1.7.0.
- **GEOMETRY_NAME:** (Starting with GDAL 2.0) 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:** 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. Default to YES.
- **SPATIAL_INDEX=YES/NO:** (Starting with GDAL 1.7.0) 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. Default to YES.
- **COMPRESS_GEOM=YES/NO:** (Starting with GDAL 1.9.0) 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=srid:** (Starting with GDAL 1.10) 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, ...]:** (Starting with GDAL 1.10.0) 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=fid_name:** (From GDAL 2.0) Name of the FID column to create. Defaults to OGC_FID.

5.83.10 Other Configuration Options

See other configure options [here](#).

5.83.11 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 behaviour is to COMMIT a transaction every 20000 inserted rows. The **-gt** argument allows explicitly setting the number of rows for each transaction. Increasing to **-gt 65536** or more ensures optimal performance while populating some table containing many hundredth thousand or million rows.

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. Starting since GDAL 1.9.0 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"
```

5.83.12 Example

- Convert 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"
```

5.83.13 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](#)

5.83.14 Links

- <http://www.sqlite.org>: Main SQLite page.
- <http://www.gaia-gis.it/spatialite/>: Spatialite extension to SQLite.
- [FDO RFC 16](#): FDO Provider for SQLite
- *RasterLite2 driver*

5.84 SUA - Tim Newport-Peace's Special Use Airspace Format

Driver short name

SUA

Driver built-in by default

This driver is built-in by default

This driver reads files describing Special Use Airspaces in the Tim Newport-Peace's .SUA format

Airspace are returned as features of a single layer, with a geometry of type Polygon and the following fields : TYPE, CLASS, TITLE, TOPS, BASE.

Airspace geometries made of arcs will be tessellated.

5.84.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

5.84.2 See Also

- [Description of .SUA format](#)

5.85 SVG - Scalable Vector Graphics

Driver short name

SVG

Build dependencies

libexpat

OGR has support for SVG reading (if GDAL is built with *expat* library support).

Currently, it will only read SVG files that are the output from Cloudmade Vector Stream Server

All coordinates are relative to the Pseudo-mercator SRS (EPSG:3857).

The driver will return 3 layers :

- points
- lines
- polygons

5.85.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

5.85.2 See Also

- [W3C SVG page](#)
- [Cloudmade vector documentation](#)

5.86 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.

5.86.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

5.86.2 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 \(rus\)](#)

5.87 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 believed that any TIGER/Line product from the 1990's should work with the reader, with the possible loss of some version specific information.

5.87.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

5.87.2 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.

5.87.2.1 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.

5.87.2.2 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.

5.87.2.3 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.

5.87.2.4 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.

5.87.2.5 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.

5.87.2.6 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.

5.87.2.7 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.

5.87.2.8 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.

5.87.2.9 EntityNames

These features are derived from type C (Geographic Entity Names) records.

These features have no associated geometry.

5.87.2.10 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.

5.87.2.11 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.

5.87.2.12 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.

5.87.2.13 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.

5.87.3 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.

5.88 TopoJSON driver

Driver short name

TopoJSON

Driver built-in by default

This driver is built-in by default

(Note: prior to GDAL 2.3, the functionality of this driver was available in the GeoJSON driver. They are now distinct drivers)

The driver can read the [TopoJSON format](#)

5.88.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

5.88.2 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

Starting with GDAL 2.3, the URL/filename/text might be prefixed with TopoJSON: to avoid any ambiguity with other drivers.

5.88.3 See Also

- [GeoJSON driver](#)
- [TopoJSON Format Specification](#)

5.89 VDV - VDV-451/VDV-452/INTREST Data Format

New in version 2.1.

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.

5.89.1 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.)*

5.89.2 Creations 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.

The following dataset creation options are available:

- **SINGLE_FILE**=YES/NO. Whether several layers should be put in the same file. If NO, the name is assumed to be a directory name. Defaults to YES.

The following layer creation options are available:

- **EXTENSION**=string. Extension used when creation files in separate layers, i.e. only for SINGLE_FILE=NO dataset creation option. Defaults to x10.
- **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. Whether checks of profile should be strict. In strict mode, unexpected layer or field names will be rejected. Defaults to NO.
- **CREATE_ALL_FIELDS**=YES/NO. Whether all fields of predefined profiles should be created at layer creation. Defaults to YES.
- **STANDARD_HEADER**=YES/NO. 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. Defaults to YES.
- **HEADER_SRC**=string: Value of the src header field. Defaults to UNKNOWN.
- **HEADER_SRC_DATE**=string: Value of the date of the src header field as DD.MM.YYYY. Defaults to current date (in GMT).

- **HEADER_SRC_TIME**=string: Value of the time of the src header field as HH.MM.SS. Defaults to current time (in GMT)
- **HEADER_CHS**=string: Value of the chs header field. Defaults to ISO8859-1.
- **HEADER_VER**=string: Value of the ver header field. Defaults to 1.4.
- **HEADER_IFV**=string: Value of the ifv header field. Defaults to 1.4.
- **HEADER_DVE**=string: Value of the dve header field. Defaults to 1.4.
- **HEADER_FFT**=string: Value of the fft header field. Defaults to '' (empty string).
- **HEADER_XXX**=string: Value of the xxx (user defined) header field.

5.89.3 Links

- [VDV-451 file format](#) (German)
- [VDV-452 data model](#) (German)
- [Austrian INTREST data format](#) (German)

5.90 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 zero or more 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.

5.90.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (/vsimem/, etc.)*

5.90.2 Open options

Starting with GDAL 2.3, the following open options can be specified (typically with the “-oo name=value” parameters of ogrinfo or ogr2ogr):

- **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.

5.90.2.1 Configuration options

(set with “-config key value” on GDAL command line utilities)

Starting with GDAL 1.9, the driver uses SQLite as a backend database when reading VFK data. By default, SQLite database is created in a directory of input VFK file (with file extension ‘.db’). Since GDAL 1.10, the user can define DB name with **OGR_VFK_DB_NAME** configuration option. If **OGR_VFK_DB_OVERWRITE=YES** configuration option is given, the driver overwrites existing SQLite database and stores data read from input VFK file into newly created DB. Since GDAL 1.11, if **OGR_VFK_DB_DELETE=YES** configuration option is given, the driver deletes backend SQLite database when closing the datasource.

Starting with GDAL 1.10, resolved geometries are stored also in backend SQLite database. It means that 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. Geometries are not stored in DB when **OGR_VFK_DB_SPATIAL=NO** configuration option is given. In this case geometries are resolved when reading data from DB on the fly.

5.90.2.2 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.

Since GDAL 1.11, 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.

5.90.3 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.

Since GDAL 2.2 also a full path to the backend SQLite database can 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 open by VFK driver instead.

5.90.4 Layer names

VFK data blocks are used as layer names.

5.90.5 Filters

5.90.5.1 Attribute filter

An internal SQL engine is used to evaluate the expression. Evaluation is done once when the attribute filter is set.

5.90.5.2 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.

5.90.6 References

- [OGR VFK Driver Implementation Issues](#)
- [Open Source Tools for VFK format \(in Czech\)](#)
- [Czech cadastral exchange data format documentation \(in Czech\)](#)

5.91 VRT – 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.

The virtual files are currently normally prepared by hand.

5.91.1 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.*)

5.91.2 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

5.91.3 Virtual File Format

The root element of the XML control file is **OGRVRTDataSource**. It has an **OGRVRTLayer** (or **OGRVRTWarpedLayer** or **OGRVRTUnionLayer** starting with GDAL 1.10.0) child for each layer in the virtual datasource, and a **Metadata** element.

A [XML schema of the OGR VRT format](#) is available. Starting with GDAL 1.11, 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`.

Metadata (optional): (GDAL >= 2.0) 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.

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): (GDAL >= 2.0) This element may list a number of open options as child elements of the form `<OOI key="key_name">value_name</OOI>`

Metadata (optional): (GDAL >= 2.0) 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. Starting with OGR 1.10, an optional **dialect** attribute can be specified on the SrcSQL element to specify which SQL “dialect” should be used : possible values are currently *OGR SQL* 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.

- Logic for GDAL >= 2.4: 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.

- Logic for GDAL < 2.4: The layer will report the FID column name only if it is also reported as a regular field. Starting with GDAL 2.0, a “name” attribute can be specified on the FID element so that the FID column name is always reported.

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.

If not provided the geometry of the source feature is copied directly.

The type of geometry encoding is indicated with the **encoding** attribute which may have the value “WKT”, “WKB” or “PointFromColumns”.

If the encoding is “WKT” or “WKB” then the **field** attribute will have the name of the field containing the WKT or WKB geometry.

If the encoding is “PointFromColumns” then the **x**, **y**, **z** and **m** attributes will have the names of the columns to be used for the X, Y, Z and M coordinates. The **z** and **m** attributes are optional (m only supported since OGR 2.1.1).

The optional **reportSrcColumn** attribute can be used to specify whether the source geometry fields (the fields set in the **field**, **x**, **y**, **z**, **m** attributes) should be reported 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.

Starting with OGR 1.11, the **GeometryField** element can be repeated as many times as necessary to create multiple geometry fields. It accepts a **name** attribute (recommended) that will be used to define the VRT geometry field name.

When **encoding** is not specified, the **field** attribute will be used to determine the corresponding geometry field name in the source layer. If neither **encoding** nor **field** are specified, it is assumed that the name of source geometry field is the value of the **name** attribute.

Starting with GDAL 2.0, the optional **nullable** attribute can be used to specify whether the geometry field is nullable. It defaults to “true”.

When several geometry fields are used, the following child elements of **GeometryField** can be defined to explicitly set the geometry type, SRS, source region, or extent.

- **GeometryType** (optional) : same syntax as OGRVRTLayer-level **GeometryType**.
- **SRS** (optional) : same syntax as OGRVRTLayer-level **LayerSRS** (note SRS vs LayerSRS)
- **SrcRegion** (optional) : same syntax as OGRVRTLayer-level **SrcRegion**
- **ExtentXMin**, **ExtentYMin**, **ExtentXMax** and **ExtentYMax** (optional) : same syntax as OGRVRTLayer-level elements of same name

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.

SrcRegion (optional, from GDAL 1.7.0) : This element is used to define an initial spatial filter for the source features. 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 polygon. 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, from GDAL 1.7.0): 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**: (GDAL >= 2.0) 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** (GDAL >= 2.0) can be used to specify whether the field is nullable. It defaults to “true”.

FeatureCount (optional, from GDAL 1.10.0) : 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 ExtentYMax (optional, from GDAL 1.10.0) : 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.

A **OGRVRTWarpedLayer** element (GDAL >= 1.10.0) 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 ExtentYMax** (optional, from GDAL 1.10.0) : 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, from GDAL 1.11) : 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.

A **OGRVRTUnionLayer** element (GDAL >= 1.10.0) 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**, GDAL >= 1.11) : 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 **ExtentYMax** (optional) : see above for the syntax

5.91.4 Example: 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>
```

5.91.5 Example: 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 (from GDAL 1.7.0) 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>
```

5.91.6 Example: Transparent spatial filtering (GDAL >= 1.7.0)

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>
```

5.91.7 Example: Reprojected layer (GDAL >= 1.10.0)

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>
```

5.91.8 Example: Union layer (GDAL >= 1.10.0)

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>
```

5.91.9 Example: SQLite/Spatialite SQL dialect (GDAL >=1.10.0)

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>
```

(continues on next page)

(continued from previous page)

```

</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>

```

5.91.10 Example: Multiple geometry fields (GDAL >= 1.11)

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>
</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>

```

(continues on next page)

(continued from previous page)

```

</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>

```

5.91.11 Other Notes

- When the *GeometryField* is “WKT” spatial filtering is applied after extracting all rows from the source data-source. 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.

5.92 Walk - Walk Spatial Data

Driver short name

Walk

Build dependencies

ODBC library

OGR optionally supports reading Walk spatial data via ODBC. Walk spatial data is a Microsoft Access database developed by Walkinfo Technologies mainly for land surveying, evaluation, planning, checking and data analysis in China.

Walk .mdb are accessed by passing the file name of the .mdb file to be accessed as the data source name. On Windows, no ODBC DSN is required. On Linux, there are problems with DSN-less connection due to incomplete or buggy implementation of this feature in the [MDB Tools](#) package, So, it is required to configure Data Source Name (DSN) if the MDB Tools driver is used (check instructions below).

OGR treats all feature tables as layers. Most geometry types should be supported (arcs and circles are translated into line segments, while other curves are currently converted into straight lines). Coordinate system information should be properly associated with layers. Currently no effort is made to preserve styles and annotations.

Currently the OGR Walk driver does not take advantage of spatial indexes for fast spatial queries.

By default, SQL statements are handled by *OGR SQL* engine. SQL commands can also be passed directly to the ODBC database engine when SQL dialect is not “OGRSQL”. In that case, the queries will deal with tables (such as “XXXXFeatures”, where XXXX is the name of a layer) instead of layers.

5.92.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

5.92.2 How to use Walk driver with unixODBC and MDB Tools (on Unix and Linux)

Refer to the similar section of the *PGeo* driver. The prefix to use for this driver is Walk:

5.93 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.

5.93.1 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.)*

5.93.2 Configuration options

- **WASP_FIELDS** : 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** : 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** : in case input has several geometry columns and the first one (default) is not the right one.
- **WASP_TOLERANCE** : specify a tolerance for line simplification of output (calls geos).
- **WASP_ADJ_TOLER** : points that are less than tolerance apart from previous point on x and on y are omitted.
- **WASP_POINT_TO_CIRCLE_RADIUS** : 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.

5.94 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).

5.94.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

5.94.2 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*

Additional optional parameters can be specified such as *TYPENAME*, *VERSION*, *MAXFEATURES* 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 it exists. Note: starting with GDAL 1.10, 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 `<OGRWFSDataSource>` element must be the first bytes of the file):

```
<OGRWFSDataSource>
  <URL>http://path/to/WFS/service[?OPTIONAL_PARAMETER1=VALUE[&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:** (OGR >= 1.10) 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`

5.94.3 Request paging

Before OGR 1.10, when reading the first feature from a layer, the whole layer content will be fetched from the server.

Some servers (such as MapServer >= 6.0) support the use of `STARTINDEX` that allows to do 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 set to ON. The page size (number of features fetched in a single request) is limited to 100 by default. 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). Starting with OGR 1.10, 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.

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`).

Starting with OGR 1.10, 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.

Starting with GDAL 2.0, the WFS driver will automatically detect if server supports paging, when requesting a WFS 2.0 server.

5.94.4 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.

Starting with GDAL 2.0, 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.

5.94.5 Layer joins

Starting with GDAL 2.0, and 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.tlid
```

or

```
SELECT * FROM my_table AS t1 JOIN another_table AS t2 ON ST_Intersects(t1.geom, t2.
↪ geom)
```

5.94.6 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().

5.94.7 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.

5.94.8 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 particular, 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.

5.94.9 Special layer : WFSLayerMetadata

(OGR >= 1.9.0)

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”)`.

5.94.10 Special layer : WFSGetCapabilities

(OGR >= 1.9.0)

A “hidden” layer called “`WFSGetCapabilities`” is filled with the raw XML result of the `GetCapabilities` request.

That layer is returned through `GetLayerByName(“WFSGetCapabilities”)`.

5.94.11 Open options (GDAL >= 2.0)

The following options are available:

- **URL=url**: URL to the WFS server endpoint. Required when using the “WFS:” string as the connection string.
- **TRUST_CAPABILITIES_BOUNDS=YES/NO**: Whether to trust layer bounds declared in GetCapabilities response, for faster GetExtent() runtime. Defaults to NO
- **EMPTY_AS_NULL=YES/NO**: (GDAL >=2.0) 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 behaviour. 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**: (GDAL >=2.0) Whether to present SRS and coordinate ordering in traditional GIS order. Defaults to YES.
- **CONSIDER_EPSG_AS_URN=YES/NO/AUTO**: (GDAL >=2.0) Whether to consider srsName like EPSG:XXXX as respecting EPSG axis order. Defaults to AUTO.
- **EXPOSE_GML_ID=YES/NO**: (GDAL >=2.0) 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. Defaults to YES.

5.94.12 Examples

Listing the types of a WFS server :

```
ogrinfo -ro WFS:http://www2.dmsolutions.ca/cgi-bin/mswfs_gmap
```

Listing the types of a WFS server whose layer structures are cached in a XML file:

```
ogrinfo -ro mswfs_gmap.xml
```

Listing the features of the popplace layer, with a spatial filter :

```
ogrinfo -ro WFS:http://www2.dmsolutions.ca/cgi-bin/mswfs_gmap popplace -spat 0 0_
↪2961766.250000 3798856.750000
```

Retrieving the features of gml:id “world.2” and “world.3” from the tows:world layer :

```
ogrinfo "WFS:http://www.tinyows.org/cgi-bin/tinyows" tows:world -ro -al -where "gml_
↪id='world.2' or gml_id='world.3'"
```

Display layer metadata (OGR >= 1.9.0):

```
ogrinfo -ro -al "WFS:http://v2.suite.opengeo.org/geoserver/ows" WFSLayerMetadata
```

5.94.13 See Also

- [OGC WFS Standard](#)
- [GML driver documentation](#)
- [OGC API - Features driver documentation](#)

5.95 XLS - MS Excel format

Driver short name

XLS

Build dependencies

libfreexl

This driver reads spreadsheets in MS Excel format. 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. (At the time of writing - with FreeXL 1.0.0a -, it means in particular that 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).

5.95.1 Configuration options

- `OGR_XLS_HEADERS = FORCE / DISABLE / 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` : By default, the driver will try to detect the data type of fields. If set to `STRING`, all fields will be of String type.

5.95.2 See Also

- [Homepage of the FreeXL library](#)

5.96 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.

5.96.1 Driver capabilities

Supports Create()

This driver supports the `GDALDriver::Create()` operation

Supports VirtualIO

This driver supports *virtual I/O operations* (*/vsimem/, etc.*)

5.96.2 Configuration options

- `OGR_XLSX_HEADERS = FORCE / DISABLE / 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_XLSX_FIELD_TYPES = STRING / 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.

5.97 X-Plane/Flightgear aeronautical data

Driver short name

XPlane

Driver built-in by default

This driver is built-in by default

The X-Plane aeronautical data is supported for read access. This data is for example used by the X-Plane and Flightgear software.

The driver is able to read the following files :

Filename	Description	Supported versions
<i>apt.dat</i>	Airport data	850, 810
<i>nav.dat</i> (or <i>earth_nav.dat</i>)	Navigation aids	810, 740
<i>fix.dat</i> (or <i>earth_fix.dat</i>)	IFR intersections	600
<i>awy.dat</i> (or <i>earth_awy.dat</i>)	Airways	640

Each file will be reported as a set of layers whose data schema is given below. The data schema is generally as close as possible to the original schema data described in the X-Plane specification. However, please note that meters (or kilometers) are always used to report heights, elevations, distances (widths, lengths), etc., even if the original data are sometimes expressed in feet or nautical miles.

Data is reported as being expressed in WGS84 datum (latitude, longitude), although the specification is not very clear on that subject.

The `OGR_XPLANE_READ_WHOLE_FILE` configuration option can be set to `FALSE` when reading a big file in regards with the available RAM (especially true for *apt.dat*). This option forces the driver not to cache features in RAM, but just to fetch the features of the current layer. Of course, this will have a negative impact on performance.

5.97.1 Driver capabilities

Supports Georeferencing

This driver supports georeferencing

Supports VirtualIO

This driver supports *virtual I/O operations (vsimem/, etc.)*

5.97.2 Examples

Converting all the layers contained in 'apt.dat' in a set of shapefiles :

```
% ogr2ogr apt_shapes apt.dat
```

Converting all the layers contained in 'apt.dat' into a PostgreSQL database :

```
% PG_USE_COPY=yes ogr2ogr -overwrite -f PostgreSQL PG:"dbname=apt" apt.dat
```

5.97.3 See Also

- [X-Plane data file definitions](#)

5.97.4 Airport data (apt.dat)

This file contains the description of elements defining airports, heliports, seapases, with their runways and taxiways, ATC frequencies, etc.

The following layers are reported :

- *APT* (Point)
- *RunwayThreshold* (Point)
- *RunwayPolygon* (Polygon)
- *WaterRunwayThreshold* (Point)
- *WaterRunwayPolygon* (Polygon)
- *Stopway* (Polygon)
- *Helipad* (Point)
- *HelipadPolygon* (Polygon)
- *TaxiwayRectangle* (Polygon)
- *Pavement* (Polygon)
- *APTBoundary* (Polygon)
- *APTLinearFeature* (Line String)
- *StartupLocation* (Point)
- *APTLightBeacon* (Point)
- *APTWindsock* (Point)
- *TaxiwaySign* (Point)
- *VASI_PAPI_WIGWAG* (Point)
- *ATCFreq* (None)

All the layers other than APT will refer to the airport thanks to the “apt_icao” column, that can serve as a foreign key.

5.97.4.1 APT layer

Main description for an airport. The position reported will be the position of the tower view point if present, otherwise the position of the first runway threshold found.

Fields:

- *apt_icao*: String (5.0). ICAO code for the airport.
- *apt_name*: String (0.0). Full name of the airport.
- *type*: Integer (1.0). Airport type : 0 for regular airport, 1 for seaplane/floatplane base, 2 for heliport (added in GDAL 1.7.0)
- *elevation_m*: Real (8.2). Elevation of the airport (in meters).

- `has_tower`: Integer (1.0). Set to 1 if the airport has a tower view point.
- `hgt_tower_m`: Real (8.2). Height of the tower view point if present.
- `tower_name`: String (32.0). Name of the tower view point if present.

5.97.4.2 RunwayThreshold layer

This layer contains the description of one threshold of a runway.

The runway itself is fully be described by its 2 thresholds, and the RunwayPolygon layer.

Note : when a runway has a displaced threshold, the threshold will be reported as 2 features : one at the non-displaced threshold position (`is_displaced=0`), and another one at the displaced threshold position (`is_displaced=1`).

Fields:

- `apt_icao`: String (5.0). ICAO code for the airport of this runway threshold.
- `rwy_num`: String (3.0). Code for the runway, such as 18, 02L, etc. . . Unique for each airport.
- `width_m`: Real (3.0). Width in meters.
- `surface`: String (0.0). Type of the surface among :
 - Asphalt
 - Concrete
 - Turf/grass
 - Dirt
 - Gravel
 - Dry lakebed
 - Water
 - Snow
 - Transparent
- `shoulder`: String (0.0). Type of the runway shoulder among :
 - None
 - Asphalt
 - Concrete
- `smoothness`: Real (4.2). Runway smoothness. Percentage between 0.00 and 1.00. 0.25 is the default value.
- `centerline_lights`: Integer (1.0). Set to 1 if the runway has centre-line lights
- `edge_lighting`: String (0.0). Type of edge lighting among :
 - None
 - Yes (when imported from V810 records)
 - LIRL : Low intensity runway lights (proposed for V90x)
 - MIRL : Medium intensity runway lights
 - HIRL : High intensity runway lights (proposed for V90x)
- `distance_remaining_signs`: Integer (1.0). Set to 1 if the runway has ‘distance remaining’ lights.

- `displaced_threshold_m`: Real (3.0). Distance between the threshold and the displaced threshold.
- `is_displaced`: Integer (1.0). Set to 1 if the position is the position of the displaced threshold.
- `stopway_length_m`: Real (3.0). Length of stopway/blastpad/over-run at the approach end of runway in meters
- `markings`: String (0.0). Runway markings for the end of the runway among :
 - None
 - Visual
 - Non-precision approach
 - Precision approach
 - UK-style non-precision
 - UK-style precision
- `approach_lighting`: String (0.0). Approach lighting for the end of the runway among :
 - None
 - ALSF-I
 - ALSF-II
 - Calvert
 - Calvert ISL Cat II and III
 - SSALR
 - SSALS (V810 records)
 - SSALF
 - SALS
 - MALSR
 - MALSF
 - MALS
 - ODALS
 - RAIL
- `touchdown_lights`: Integer (1.0). Set to 1 if the runway has touchdown-zone lights (TDZL)
- `REIL`: String (0.0). Runway End Identifier Lights (REIL) among :
 - None
 - Omni-directional
 - Unidirectionnal
- `length_m`: Real (5.0). (Computed field). Length in meters between the 2 thresholds at both ends of the runway. The displaced thresholds are not taken into account in this computation.
- `true_heading_deg`: Real (6.2). (Computed field). True heading in degree at the approach of the end of the runway.

5.97.4.3 RunwayPolygon layer

This layer contains the rectangular shape of a runway. It is computed from the runway threshold information. When not specified, the meaning of the fields is the same as the *RunwayThreshold* layer. Fields:

- apt_icao: String (5.0)
- rwy_num1: String (3.0). Code for first runway threshold. For example 20L.
- rwy_num2: String (3.0). Code for the second the runway threshold. For example 02R.
- width_m: Real (3.0)
- surface: String (0.0)
- shoulder: String (0.0)
- smoothness: Real (4.2)
- centerline_lights: Integer (1.0)
- edge_lighting: String (0.0)
- distance_remaining_signs: Integer (1.0)
- length_m: Real (5.0)
- true_heading_deg: Real (6.2). True heading from the first runway to the second runway.

5.97.4.4 WaterRunwayThreshold (Point)

Fields:

- apt_icao: String (5.0)
- rwy_num: String (3.0). Code for the runway, such as 18. Unique for each airport.
- width_m: Real (3.0)
- has_buoys: Integer (1.0). Set to 1 if the runway should be marked with buoys bobbing in the water
- length_m: Real (5.0). (Computed field) Length between the two ends of the water runway.
- true_heading_deg: Real (6.2). (Computed field). True heading in degree at the approach of the end of the runway.

5.97.4.5 WaterRunwayPolygon (Polygon)

This layer contains the rectangular shape of a water runway. It is computed from the water runway threshold information. Fields:

- apt_icao: String (5.0)
- rwy_num1: String (3.0)
- rwy_num2: String (3.0)
- width_m: Real (3.0)
- has_buoys: Integer (1.0)
- length_m: Real (5.0)
- true_heading_deg: Real (6.2)

5.97.4.6 Stopway layer (Polygon)

(Starting with GDAL 1.7.0) This layer contains the rectangular shape of a stopway/blastpad/over-run that may be found at the beginning of a runway. It is part of the tarmac but not intended to be used for normal operations. It is computed from the runway stopway/blastpad/over-run length information and only present when this length is non zero. When not specified, the meaning of the fields is the same as the *RunwayThreshold* layer. Fields:

- apt_icao: String (5.0)
- rwy_num: String (3.0).
- width_m: Real (3.0)
- length_m: Real (5.0) : Length of stopway/blastpad/over-run at the approach end of runway in meters.

5.97.4.7 Helipad (Point)

This layer contains the center of a helipad. Fields:

- apt_icao: String (5.0)
- helipad_name: String (5.0). Name of the helipad in the format “Hxx”. Unique for each airport.
- true_heading_deg: Real (6.2)
- length_m: Real (5.0)
- width_m: Real (3.0)
- surface: String (0.0). See above runway *surface* codes.
- markings: String (0.0). See above runway *markings* codes.
- shoulder: String (0.0). See above runway *shoulder* codes.
- smoothness: Real (4.2). See above runway *smoothness* description.
- edge_lighting: String (0.0). Helipad edge lighting among :
 - None
 - Yes (V810 records)
 - Yellow
 - White (proposed for V90x)
 - Red (V810 records)

5.97.4.8 HelipadPolygon (Polygon)

This layer contains the rectangular shape of a helipad. The fields are identical to the *Helipad* layer.

5.97.4.9 TaxiwayRectangle (Polygon) - V810 record

This layer contains the rectangular shape of a taxiway. Fields:

- apt_icao: String (5.0)
- true_heading_deg: Real (6.2)
- length_m: Real (5.0)
- width_m: Real (3.0)
- surface: String (0.0). See above runway *surface* codes.
- smoothness: Real (4.2). See above runway *smoothness* description.
- edge_lighting: Integer (1.0). Set to 1 if the taxiway has edge lighting.

5.97.4.10 Pavement (Polygon)

This layer contains polygonal chunks of pavement for taxiways and aprons. The polygons may include holes.

The source file may contain Bezier curves as sides of the polygon. Due to the lack of support for such geometry into OGR Simple Feature model, Bezier curves are discretized into linear pieces.

Fields:

- apt_icao: String (5.0)
- name: String (0.0)
- surface: String (0.0). See above runway *surface* codes.
- smoothness: Real (4.2). See above runway *smoothness* description.
- texture_heading: Real (6.2). Pavement texture grain direction in true degrees

5.97.4.11 APTBoundary (Polygon)

This layer contains the boundary of the airport. There is at the maximum one such feature per airport. The polygon may include holes. Bezier curves are discretized into linear pieces.

Fields:

- apt_icao: String (5.0)
- name: String (0.0)

5.97.4.12 APTLinearFeature (Line String)

This layer contains linear features. Bezier curves are discretized into linear pieces.

Fields:

- apt_icao: String (5.0)
- name: String (0.0)

5.97.4.13 StartupLocation (Point)

Define gate positions, ramp locations etc.

Fields:

- apt_icao: String (5.0)
- name: String (0.0)
- true_heading_deg: Real (6.2)

5.97.4.14 APTLightBeacon (Point)

Define airport light beacons.

Fields:

- apt_icao: String (5.0)
- name: String (0.0)
- color: String (0.0). Color of the light beacon among :
 - None
 - White-green: land airport
 - White-yellow: seaplane base
 - Green-yellow-white: heliports
 - White-white-green: military field

5.97.4.15 APTWindsock (Point)

Define airport windsocks.

Fields:

- apt_icao: String (5.0)
- name: String (0.0)
- is_illuminated: Integer (1.0)

5.97.4.16 TaxiwaySign (Point)

Define airport taxiway signs.

Fields:

- apt_icao: String (5.0)
- text: String (0.0). This is somehow encoded into a specific format. See X-Plane [specification](#) (pages 13 and 14) for more details.
- true_heading_deg: Real (6.2)
- size: Integer (1.0). From 1 to 5. See X-Plane [specification](#) for more details.

5.97.4.17 VASI_PAPI_WIGWAG (Point)

Define a VASI, PAPI or Wig-Wag. For PAPIs and Wig-Wags, the coordinate is the centre of the display. For VASIs, this is the mid point between the two VASI light units.

Fields:

- apt_icao: String (5.0)
- rwy_num: String (3.0). Foreign key to the rwy_num field of the *RunwayThreshold* layer.
- type: String (0.0). Type among :
 - VASI
 - PAPI Left
 - PAPI Right
 - Space Shuttle PAPI
 - Tri-colour VASI
 - Wig-Wag lights
- true_heading_deg: Real (6.2)
- visual_glide_deg: Real (4.2)

5.97.4.18 ATCFreq (None)

Define an airport ATC frequency. Note that this layer has no geometry.

Fields:

- apt_icao: String (5.0)
- atc_type: String (4.0). Type of the frequency among (derived from the record type number) :
 - ATIS : AWOS (Automatic Weather Observation System), ASOS (Automatic Surface Observation System) or ATIS (Automated Terminal Information System)
 - CTAF : Unicom or CTAF (USA), radio (UK)
 - CLD : Clearance delivery (CLD)
 - GND : Ground
 - TWR : Tower
 - APP : Approach
 - DEP : Departure
- freq_name: String (0.0). Name of the ATC frequency. This is often an abbreviation (such as GND for “Ground”).
- freq_mhz: Real (7.3). Frequency in MHz.

5.97.5 Navigation aids (nav.dat)

This file contains the description of various navigation aids beacons.

The following layers are reported :

- *ILS* (Point)
- *VOR* (Point)
- *NDB* (Point)
- *GS* (Point)
- *Marker* (Point)
- *DME* (Point)
- *DMEILS* (Point)

5.97.5.1 ILS (Point)

Localizer that is part of a full ILS, or Stand-alone localizer (LOC), also including a LDA (Landing Directional Aid) or SDF (Simplified Directional Facility).

Fields :

- `navaid_id`: String (4.0). Identification of nav-aid. *NOT* unique.
- `apt_icao`: String (5.0). Foreign key to the `apt_icao` field of the *RunwayThreshold* layer.
- `rwy_num`: String (3.0). Foreign key to the `rwy_num` field of the *RunwayThreshold* layer.
- `subtype`: String (10.0). Sub-type among :
 - ILS-cat-I
 - ILS-cat-II
 - ILS-cat-III
 - LOC
 - LDA
 - SDF
 - IGS
 - LDA-GS
- `elevation_m`: Real (8.2). Elevation of nav-aid in meters.
- `freq_mhz`: Real (7.3). Frequency of nav-aid in MHz.
- `range_km`: Real (7.3). Range of nav-aid in km.
- `true_heading_deg`: Real (6.2). True heading of the localizer in degree.

5.97.5.2 VOR (Point)

Navaid of type VOR, VORTAC or VOR-DME.

Fields :

- `navaid_id`: String (4.0). Identification of nav-aid. *NOT* unique.
- `navaid_name`: String (0.0)
- `subtype`: String (10.0). Among VOR, VORTAC or VOR-DME
- `elevation_m`: Real (8.2)
- `freq_mhz`: Real (7.3)
- `range_km`: Real (7.3)
- `slaved_variation_deg`: Real (6.2). Indicates the slaved variation of a VOR/VORTAC in degrees.

5.97.5.3 NDB (Point)

Fields :

- `navaid_id`: String (4.0). Identification of nav-aid. *NOT* unique.
- `navaid_name`: String (0.0)
- `subtype`: String (10.0). Among NDB, LOM, NDB-DME.
- `elevation_m`: Real (8.2)
- `freq_khz`: Real (7.3). Frenquency in **kHz**
- `range_km`: Real (7.3)

5.97.5.4 GS - Glideslope (Point)

Glideslope nav-aid.

Fields :

- `navaid_id`: String (4.0). Identification of nav-aid. *NOT* unique.
- `apt_icao`: String (5.0). Foreign key to the `apt_icao` field of the *RunwayThreshold* layer.
- `rwy_num`: String (3.0). Foreign key to the `rwy_num` field of the *RunwayThreshold* layer.
- `elevation_m`: Real (8.2)
- `freq_mhz`: Real (7.3)
- `range_km`: Real (7.3)
- `true_heading_deg`: Real (6.2). True heading of the glideslope in degree.
- `glide_slope`: Real (6.2). Glide-slope angle in degree (typically 3 degree)

5.97.5.5 Marker - ILS marker beacons. (Point)

Nav-aids of type Outer Marker (OM), Middle Marker (MM) or Inner Marker (IM).

Fields:

- apt_icao: String (5.0). Foreign key to the apt_icao field of the *RunwayThreshold* layer.
- rwy_num: String (3.0). Foreign key to the rwy_num field of the *RunwayThreshold* layer.
- subtype: String (10.0). Among OM, MM or IM.
- elevation_m: Real (8.2)
- true_heading_deg: Real (6.2). True heading of the glideslope in degree.

5.97.5.6 DME (Point)

DME, including the DME element of an VORTAC, VOR-DME or NDB-DME.

Fields:

- navaid_id: String (4.0). Identification of nav-aid. *NOT* unique.
- navaid_name: String (0.0)
- subtype: String (10.0). Among VORTAC, VOR-DME, TACAN or NDB-DME
- elevation_m: Real (8.2)
- freq_mhz: Real (7.3)
- range_km: Real (7.3)
- bias_km: Real (6.2). This bias must be subtracted from the calculated distance to the DME to give the desired cockpit reading

5.97.5.7 DMEILS (Point)

DME element of an ILS.

Fields:

- navaid_id: String (4.0). Identification of nav-aid. *NOT* unique.
- apt_icao: String (5.0). Foreign key to the apt_icao field of the *RunwayThreshold* layer.
- rwy_num: String (3.0). Foreign key to the rwy_num field of the *RunwayThreshold* layer.
- elevation_m: Real (8.2)
- freq_mhz: Real (7.3)
- range_km: Real (7.3)
- bias_km: Real (6.2). This bias must be subtracted from the calculated distance to the DME to give the desired cockpit reading

5.97.6 IFR intersections (fix.dat)

This file contain IFR intersections (often referred to as “fixes”).

The following layer is reported :

- *FIX* (Point)

5.97.6.1 FIX (Point)

Fields:

- *fix_name*: String (5.0). Intersection name. *NOT* unique.

5.97.7 Airways (awy.dat)

This file contains the description of airway segments.

The following layers are reported :

- *AirwaySegment* (Line String)
- *AirwayIntersection* (Point)

5.97.7.1 AirwaySegment (Line String)

Fields:

- *segment_name*: String (0.0)
- *point1_name*: String (0.0) : Name of intersection or nav-aid at the beginning of this segment
- *point2_name*: String (0.0) : Name of intersection or nav-aid at the beginning of this segment
- *is_high*: Integer (1.0) : Set to 1 if this is a “High” airway.
- *base_FL*: Integer (3.0) : Flight level (hundreds of feet) of the base of the airway.
- *top_FL*: Integer (3.0) : Flight level (hundreds of feet) of the top of the airway.

5.97.7.2 AirwayIntersection (Point)

Fields:

- *name*: String (0.0) : Name of intersection or nav-aid

USER ORIENTED DOCUMENTATION

6.1 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.

6.1.1 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.

Note that the GDAL dataset, and raster band data model is loosely based on the OpenGIS Grid Coverages specification.

6.1.2 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 (i.e. Transverse Mercator).
- A list of projection parameters (i.e. 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.

6.1.3 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).

6.1.4 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.

6.1.5 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 NO-DATA_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.

6.1.5.1 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 advertize 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, OGD, PDF, PostGISRaster, Rasterlite, RPFTOC, RS2, TileDB, WCS, and WMS.

6.1.5.2 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_Byte`.
- **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.
- **PIXELTYPE**: This may appear on a `GDT_Byte` band (or the corresponding dataset) and have the value `SIGNED-BYTE` 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.

6.1.5.3 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.

6.1.5.4 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

6.1.5.5 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.

6.1.6 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 Byte, UInt16, Int16, UInt32, Int32, Float32, Float64, and the complex types CInt16, CInt32, CFloat32, and CFloat64.
- 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 (i.e. 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*: the default, nothing is known.
 - *GCI_GrayIndex*: this is an independent gray-scale image
 - *GCI_PaletteIndex*: this raster acts as an index into a color table
 - *GCI_RedBand*: this raster is the red portion of an RGB or RGBA image

- GCI_GreenBand: this raster is the green portion of an RGB or RGBA image
 - GCI_BlueBand: this raster is the blue portion of an RGB or RGBA image
 - GCI_AlphaBand: this raster is the alpha portion of an RGBA image
 - GCI_HueBand: this raster is the hue of an HLS image
 - GCI_SaturationBand: this raster is the saturation of an HLS image
 - GCI_LightnessBand: this raster is the hue of an HLS image
 - GCI_CyanBand: this band is the cyan portion of a CMY or CMYK image
 - GCI_MagentaBand: this band is the magenta portion of a CMY or CMYK image
 - GCI_YellowBand: this band is the yellow portion of a CMY or CMYK image
 - GCI_BlackBand: this band is the black portion of a CMYK image.
- A color table, described in more detail later.
 - Knowledge of reduced resolution overviews (pyramids) if available.

6.1.7 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;

    /* 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.

6.1.8 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.

6.2 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*.

6.2.1 Group

A *GDALGroup* (modelling a [HDF5 Group](#)) is a named container of *GDALAttribute*, *GDALMDArray* or other GDAL-Group. Hence *GDALGroup* can describe a hierarchy of objects.

6.2.2 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)

6.2.3 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 $\text{unscaled_value} = \text{offset} + \text{scale} * \text{raw_value}$
- Scale, such that $\text{unscaled_value} = \text{offset} + \text{scale} * \text{raw_value}$

6.2.4 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.

6.2.5 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.

6.2.6 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.

6.2.7 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.

6.2.8 Applications

The following applications can be used to inspect and manipulate multidimensional datasets:

- `gdalmdiminfo`
- `gdalmdimtranslate`

6.3 Vector Data Model

This document is intended to document the OGR classes. The OGR classes are intended to be generic (not specific to OLE DB or COM or Windows) but are used as a foundation for implementing OLE DB Provider support, as well as client side support for SFCOM. It is intended that these same OGR classes could be used by an implementation of SFCORBA for instance or used directly by C++ programs wanting to use an OpenGIS simple features inspired API.

Because OGR is modeled on the OpenGIS simple features data model, it is very helpful to review the SFCOM, or other simple features interface specifications which can be retrieved from the Open Geospatial Consortium web site. Data types, and method names are modeled on those from the interface specifications.

6.3.1 Class Overview

- Geometry (*ogr_geometry.h*): The geometry classes (*OGRGeometry*, etc) encapsulate the OpenGIS model vector data as well as providing 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 geometry and a set of attributes.
- 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 (*ogr_sfrmts.h*): *OGRLayer* is an abstract base class represent a layer of features in an *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, opening *GDALDataset* objects. All available drivers are managed by the *GDALDriverManager*.

6.3.2 Geometry

The geometry classes are represent various kinds of vector geometry. All the geometry classes derived from *OGRGeometry* which defines the common services of all geometries. Types of geometry include *OGRPoint*, *OGRLineString*, *OGRPolygon*, *OGRGeometryCollection*, *OGRMultiPolygon*, *OGRMultiPoint*, and *OGRMultiLineString*.

GDAL 2.0 extends those geometry type with non-linear geometries with the *OGRCircularString*, *OGRCompoundCurve*, *OGRCurvePolygon*, *OGRMultiCurve* and *OGRMultiSurface* classes.

Additional intermediate abstract base classes contain functionality that could eventually be implemented by other 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, and well known binary format data into geometries. These are predefined ASCII and binary formats for representing all the types of simple features geometries.

In a manner based on the geometry object in SFCOM, 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.

Many of the spatial analysis methods (such as computing overlaps and so forth) are not implemented at this time for *OGRGeometry*.

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.

6.3.2.1 Compatibility issues with GDAL 2.0 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.

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.

6.3.3 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. There is a tutorial available describing how to use the *OGRSpatialReference* class.

6.3.4 Feature / Feature Definition

The *OGRGeometry* captures the geometry of a vector feature ... the spatial position/region of a feature. The *OGRFeature* contains this geometry, and adds feature attributes, feature id, and a feature class identifier. Starting with OGR 1.11, several geometries can be associated to a *OGRFeature*.

The set of attributes, 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 (GDAL 2.0 or later); 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 feature class also contains an indicator of the types of geometry allowed for that feature class (returned as an *OGRwkbGeometryType* from *OGRFeatureDefn::GetGeomType()*). If this is *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.

Starting with OGR 1.11, several geometry fields can be associated to a feature class. 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).

6.3.5 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, normally a file format. In SFCOM and other table based simple features implementation an *OGRLayer* represents a spatial table.

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).

One flaw in the current OGR architecture is that the spatial filter is 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. This aspect may be revised in the future to introduce an *OGRLayerView* class or something similar.

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.

6.3.6 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) 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.

6.3.7 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.

6.4 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.

6.4.1 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.

6.4.1.1 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 behaviour. 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.

6.4.1.2 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.

6.4.2 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.

6.4.2.1 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.

6.4.3 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 vertexes 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()*

6.5 OGR SQL dialect and SQLITE SQL dialect

6.5.1 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 issue with driver specific SQL support.

Since GDAL/OGR 1.10, 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.

NOTE: OGR SQL has been reimplemented for GDAL/OGR 1.8.0. Many features discussed below, notably arithmetic expressions, and expressions in the field list, were not support in GDAL/OGR 1.7.x and earlier. See RFC 28 for details of the new features in GDAL/OGR 1.8.0.

6.5.1.1 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 EAS_ID and PROP_VALUE attributes. Any other attributes would be discarded.

```
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 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 separate 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.

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 (a count of instances), AVG (numerical average), SUM (numerical sum), MIN (lexical or numerical minimum), and MAX (lexical or numerical maximum). 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) 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
```

Note: prior to OGR 1.9.0, null values were counted in COUNT(column_name) or COUNT(DISTINCT column_name), which was not conformant with the SQL standard. Since OGR 1.9.0, only non-null values are counted.

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. Starting with GDAL 2.0, 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

Starting with OGR 1.8.2, 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'
```

Starting with OGR 2.0, 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

Starting with GDAL 1.6.0, 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 (GDAL >= 2.0)
- character(field_length). By default, field_length=1.
- float(field_length)
- numeric(field_length, field_precision)
- smallint(field_length) : 16 bit signed integer (GDAL >= 2.0)
- integer(field_length)
- bigint(field_length), 64 bit integer, extension to SQL92 (GDAL >= 2.0)
- 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.

Starting with OGR 1.11, 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

Starting with GDAL 2.0, 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 behaviour is wished in GDAL 3.1, the `OGR_SQL_LIKE_AS_ILIKE` can be set to YES.

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 of the field values. 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
```

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

Starting with GDAL 2.2, 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
```

Before GDAL 2.0, the expression after `ON` should necessarily be of the form “`{primary_table}.{field_name} = {secondary_table}.{field_name}`”, and in that order. Starting with GDAL 2.0, it is 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.

6.5.1.2 SPECIAL FIELDS

The OGR SQL query processor treats some of the attributes of the features as built-in special fields can be used in the SQL statements likewise the other fields. These fields can be placed in the select list, the WHERE clause and the ORDER BY clause respectively. The special field will not be included in the result by default but it may be explicitly included by adding it to the select list. When accessing the field values the special fields will take precedence over the other fields with the same names in the data source.

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`. The field wildcard expansions will not include the feature id, but it may be explicitly included using a syntax like:

```
SELECT FID, * 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%'
```

6.5.1.3 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 they will accelerate is a simple “field = value” query.

6.5.1.4 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
```

6.5.1.5 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 `OLCAAlterFieldDefn` capability. -“`ALTER TABLE tablename ALTER [COLUMN] columnname TYPE columntype`” to change the type of an existing field. Supported if the layer declares the `OLCAAlterFieldDefn` 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
```

6.5.1.6 DROP TABLE

The OGR SQL DROP TABLE command can be used to delete a table. This is only supported on datasources that declare the ODSDeleteLayer capability.

```
DROP TABLE nation
```

6.5.1.7 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 data-source is destroyed may result in a crash.

6.5.1.8 Non-OGR SQL

All OGR drivers for database systems: *MySQL*, *PostgreSQL / PostGIS*, *Oracle Spatial*, *SQLite / Spatialite RDBMS*, *ODBC RDBMS*, *ESRI Personal GeoDatabase*, 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. Only the result of SQL WHERE statements will be returned as layers.

6.5.2 SQL SQLite dialect

Since GDAL/OGR 1.10, 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 (≥ 3.6), 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. It is available through the `GDALDataset::ExecuteSQL()` method by specifying the `pszDialect` to “SQLITE”. For the *ogrinfo* or *ogr2ogr* utility, you must specify the “-dialect SQLITE” option.

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.

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

6.5.2.1 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)
```

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 can also be used, spatial functions, etc. ...

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.

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

The GEOMETRY special field represents the geometry of the feature returned by OGRFeature::GetGeometryRef(). It can be explicitly specified in the result column list of a SELECT, and is 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 will be the name of the geometry column of the underlying OGR layer.

```
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))
```

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%'
```

Spatialite SQL functions

When GDAL/OGR is build with support for the [Spatialite](http://www.gaia-gis.it/spatialite/) library, a lot of [extra SQL functions](http://www.gaia-gis.it/gaia-sins/spatialite-sql-4.3.0.html), 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
```

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('libspatialite.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')
```

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

Starting with OGR 2.0, 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'
```

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 (2.342878767069653 48.85661793020374)
```

```
ogrinfo cities.csv -dialect sqlite -sql "SELECT *, ogr_geocode(city, 'country') AS_  
↪country, ST_Centroid(ogr_geocode(city)) FROM cities"
```

returns:

```
OGRFeature(SELECT):0  
id (Real) = 1  
city (String) = Paris  
country (String) = France métropolitaine  
POINT (2.342878767069653 48.85661793020374)  
  
OGRFeature(SELECT):1  
id (Real) = 2  
city (String) = London  
country (String) = United Kingdom  
POINT (-0.109369427546499 51.500506667319407)  
  
OGRFeature(SELECT):2  
id (Real) = 3  
city (String) = Rennes  
country (String) = France métropolitaine  
POINT (-1.68185153381778 48.111663929761093)  
  
OGRFeature(SELECT):3  
id (Real) = 4  
city (String) = Strasbourg  
country (String) = France métropolitaine  
POINT (7.767762859150757 48.571233274141846)  
  
OGRFeature(SELECT):4  
id (Real) = 5  
city (String) = New York  
country (String) = United States of America  
POINT (-73.938140243499049 40.663799577449979)  
  
OGRFeature(SELECT):5  
id (Real) = 6  
city (String) = Berlin
```

(continues on next page)

(continued from previous page)

```

country (String) = Deutschland
POINT (13.402306623451983 52.501470321410636)

OGRFeature(SELECT):6
id (Real) = 7
city (String) = Beijing
POINT (116.391195 39.9064702)

OGRFeature(SELECT):7
id (Real) = 8
city (String) = Brasilia
country (String) = Brasil
POINT (-52.830435216371839 -10.828214867369699)

OGRFeature(SELECT):8
id (Real) = 9
city (String) = Moscow
country (String) = Российская Федерация
POINT (37.367988106866868 55.556208255649558)

```

`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)

```

6.6 GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...

6.6.1 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, 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
```

6.6.2 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/poly.zip

# ogrinfo a shapefile in a zip file on an ftp:
ogrinfo -ro -al -so /vsizip//vsicurl/ftp://user:password@example.com/foldername/file.
↳ zip/example.shp
```

6.6.3 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`.

Notable exceptions are the netCDF, HDF4 and HDF5 drivers.

6.6.4 /vsizip/ (.zip archives)

/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 `:cpp:func:`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.

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
```

.kmz, .ods and .xlsx extensions are also detected as valid extensions for zip-compatible archives.

Starting with GDAL 2.2, 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.

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);
```

Starting with GDAL 2.4, 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.

Read and write operations cannot be interleaved. The new zip must be closed before being re-opened in read mode.

6.6.5 /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.

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
```

VSISatL() 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.

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 (the creation of that file can be disabled by setting the `CPL_VSIL_GZIP_WRITE_PROPERTIES` configuration option to NO).

Write capabilities are also available, but read and write operations cannot be interleaved.

Starting with GDAL 2.4, 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.

6.6.6 /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 `/vsitar/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
```

Starting with GDAL 2.2, 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.

6.6.7 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-file systems 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.

6.6.7.1 /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
```

Starting with GDAL 2.3, 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 behaviour of the `CPL_VSIL_CURL_USE_HEAD` configuration option.
- **max_retry=number**: default to 0. Setting this option overrides the behaviour of the `GDAL_HTTP_MAX_RETRY` configuration option.
- **retry_delay=number_in_seconds**: default to 30. Setting this option overrides the behaviour of the `GDAL_HTTP_RETRY_DELAY` configuration option.
- **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.

Partial downloads (requires the HTTP server to support random reading) are done with a 16 KB granularity by default. Starting with GDAL 2.3, 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 2 MB to improve download performance. Starting with GDAL 2.3, 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`, `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.

Starting with GDAL 2.1.3, 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).

Starting with GDAL 2.3, 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.

Starting with GDAL 2.3, 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.

More generally options of `CPLHTTPFetch()` available through configuration options are available.

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.

In addition, a global least-recently-used 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, during the life-time of the process or until `VSICurlClearCache()` is called. Starting with GDAL 2.3, the size of this global LRU cache can be modified by setting the configuration option `CPL_VSIL_CURL_CACHE_SIZE` (in bytes).

Starting with GDAL 2.3, 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.

Starting with GDAL 2.1, `/vsicurl/` will try to query directly redirected URLs to Amazon S3 signed URLs during their validity period, so as to minimize round-trips. This behaviour can be disabled by setting the configuration option `CPL_VSIL_CURL_USE_S3_REDIRECT` to `NO`.

`VSISizeL()` 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.

6.6.7.2 `/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`, `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.

Starting with GDAL 2.1.3, 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).

`VSISizeL()` 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).

6.6.7.3 `/vsi3/` (AWS S3 files: random reading)

`/vsi3/` 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). Deletion of files with `VSIUnlink()` is also supported. Starting with GDAL 2.3, creation of directories with `VSIMkdir()` and deletion of (empty) directories with `VSIUnlink()` are also possible.

Recognized filenames are of the form `/vsi3/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.

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. Available since GDAL 2.3

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. Starting with GDAL 2.3, 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 (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. If none of the above method succeeds, instance profile credentials will be retrieved when GDAL is used on EC2 instances.

The `AWS_REGION` (or `AWS_DEFAULT_REGION` starting with GDAL 2.3) configuration option may be set to one of the supported S3 regions and defaults to `us-east-1`.

Starting with GDAL 2.2, the `AWS_REQUEST_PAYER` configuration option may be set to “requester” to facilitate use with Requester Pays buckets.

The `AWS_S3_ENDPOINT` configuration option defaults to `s3.amazonaws.com`.

The `AWS_HTTPS` configuration option defaults to `YES`.

The `AWS_VIRTUAL_HOSTING` configuration option defaults to `TRUE`. This allows you to configure the two ways to access the buckets, see Bucket and Host Name for more details. - `TRUE` value, identifies the bucket via a virtual bucket host name, e.g.: `mybucket.cname.domain.com` - `FALSE` value, identifies the bucket as the top-level directory in the URI, e.g.: `cname.domain.com/mybucket`

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 2.4, when listing a directory, files with `GLACIER` storage class are ignored unless the `CPL_VSIL_CURL_IGNORE_GLACIER_STORAGE` configuration option is set to `NO`.

Since GDAL 3.1, the `Rename()` operation is supported (first doing a copy of the original file and then deleting it).

New in version 2.1.

6.6.7.4 `/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/`

New in version 2.1.

6.6.7.5 /vsigs/ (Google Cloud Storage files: random reading)

/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.

Starting with GDAL 2.3, 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). 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.

Several authentication methods are possible, and are attempted in the following order:

1. The `GS_SECRET_ACCESS_KEY` and `GS_ACCESS_KEY_ID` configuration options can be set for AWS-style authentication
2. 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 XXXXXXXXXX” line.
3. (GDAL >= 2.3) 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)
4. (GDAL >= 2.3) The `GOOGLE_APPLICATION_CREDENTIALS` configuration option can be set to point to a JSON file containing OAuth2 service account credentials, 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.
5. (GDAL >= 2.3) 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.
6. (GDAL >= 2.3) 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.
7. (GDAL >= 2.3) 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).

New in version 2.2.

6.6.7.6 `/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/`

New in version 2.2.

6.6.7.7 `/vsiaz/` (Microsoft Azure Blob files: random reading)

`/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`.

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). 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.

Two authentication methods are possible, and are attempted in the following order:

1. The `AZURE_STORAGE_CONNECTION_STRING` configuration option, given in the access key section of the administration interface. It contains both the account name and a secret key.
2. The `AZURE_STORAGE_ACCOUNT` and `AZURE_STORAGE_ACCESS_KEY` configuration options pointing respectively to the account name and a secret key.

Since GDAL 3.1, the `Rename()` operation is supported (first doing a copy of the original file and then deleting it).

New in version 2.3.

6.6.7.8 /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/](#)

New in version 2.3.

6.6.7.9 /vsioss/ (Alibaba Cloud OSS files: random reading)

/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). 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 `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. The default is `oss-us-east-1.aliyuncs.com`. 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 `VSIOS_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.

New in version 2.3.

6.6.7.10 /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/](#)

New in version 2.3.

6.6.7.11 /vswift/ (OpenStack Swift Object Storage: random reading)

/vswift/ 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). Deletion of files with *VSIUnlink()* is also supported. Creation of directories with *VSIMkdir()* and deletion of (empty) directories with *VSIUnlink()* are also possible.

Recognized filenames are of the form /vswift/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.

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*

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.

New in version 2.3.

6.6.7.12 /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 */vsiaws/*

New in version 2.3.

6.6.7.13 /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 (`--with-java`) and HDFS support (`--with-hdfs`). 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)*

Recognized filenames are of the form /vsihdfs/hdfsUri where hdfsUri is a valid HDFS URI.

Examples:

```
/vsihdfs/file:/tmp/my.tif  (a local file accessed through HDFS)
/vsihdfs/hdfs:/hadoop/my.tif  (a file stored in HDFS)
```

New in version 2.4.

6.6.7.14 /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). Deletion of files with *VSIUnlink()* is also supported. Creation of directories with *VSIMkdir()* and deletion of (empty) directories with *VSIrmkdir()* 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 = int_value: Replication value used when creating a file
- WEBHDFS_PERMISSION = int_value: 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)

New in version 2.4.

6.6.8 /vsistdin/ (standard input streaming)

/vsistdin/ is a file handler that allows reading from the standard input stream.

The filename syntax must be only /vsistdin/.

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.

6.6.9 /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()*.

6.6.10 /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).

6.6.11 /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 `VSISStatL()`, `VSIFOpenL()` and operations based on the file handle returned by `VSIFOpenL()` operate properly.

6.6.12 /vsisparsed/ (sparse files)

The `/vsisparsed/` 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 `/vsisparsed/` 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>
    <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.

6.6.13 File caching

This is not a proper virtual file system handler, but a C function that 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.

The `VSICachedFile` class only handles read operations at that time, and will error out on write operations.

This is done with the `VSICreateCachedFile()` function, 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).

6.6.14 /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.

6.7 Feature Style Specification

Version 0.016 - 2018-12-03

6.7.1 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*.

6.7.1.1 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.

6.7.1.2 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.

6.7.1.3 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.

6.7.1.4 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.

6.7.1.5 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.

6.7.2 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.

6.7.2.1 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:	<code>PEN(c:#FF0000,w:5px)</code>
lake:	<code>BRUSH(fc:#0000FF);PEN(c:#000000)</code>
campsite:	<code>SYMBOL(c:#00FF00,id:"points.sym-45,ogr-sym-7")</code>
label:	<code>LABEL(f:"Times New Roman",s:12pt,t:{text_attribute})</code>

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.

6.7.2.2 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.

6.7.2.3 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:

param_name	Description
c	Pen 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: 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 line
id	<p>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 well 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.</p> <p>Here is the current list of OGR pen ids (this could grow over time):</p> <ul style="list-style-type: none"> 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) <p>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"</p> <p>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</p>

6.7. Feature Style Specification

MapInfo MIF to TAB translation without any loss.)

Examples:



PEN(c:#00FF00,id:"ogr-pen-0") - simple solid line

6.7.2.4 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:












param_name	Description
fc	<p>Brush ForeColor, expressed in hexadecimal (#RRGGBB[AA]). Used for painting the brush pattern itself.</p> <p>[AA] the last 2 digits define the alpha channel value, with 0 being transparent and FF being opaque. The default for [AA] is FF (opaque).</p> <p>Suggested default: 50% grey (c:#808080)</p> <p>Example: BRUSH(fc:#FF0000)</p>
bc	<p>Brush BackColor, expressed in hexadecimal (#RRGGBB[AA]). Used for painting the area behind the brush pattern.</p> <p>[AA] the last 2 digits define the alpha channel value, with 0 being transparent and FF being opaque. The default for [AA] is FF (opaque)</p> <p>Suggested default: transparent (c:#FFFFFF00)</p> <p>Example: BRUSH(fc:#FF0000,bc:#FEEEDD)</p>
id	<p>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.</p> <p>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.</p> <div><p>ogr-brush-0 ogr-brush-1 ogr-brush-2 ogr-brush-3</p><p>ogr-brush-4 ogr-brush-5 ogr-brush-6 ogr-brush-7</p></div> <p>BRUSH(fc:#808080,bc:#e0e0e0,id:"ogr-brush-n")</p> <p>Here is the current list of OGR brush ids (this could grow over time):</p> <ul style="list-style-type: none">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 hatchogr-brush-3: vertical hatchogr-brush-4: top-left to bottom-right diagonal hatchogr-brush-5: bottom-left to top-right diagonal hatchogr-brush-6: cross hatch
6.7. Feature Style Specification	<ul style="list-style-type: none">ogr-brush-6: cross hatch

6.7.2.5 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:

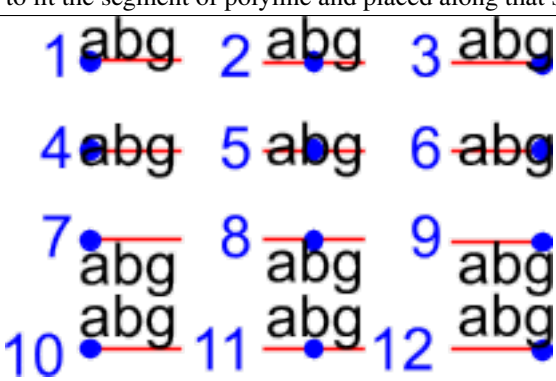
param_name	Description
id	<p>Symbol Name or Id - Comma-delimited list of symbol names or ids. The names in the comma-delimited list of ids are scanned until one is recognized by the target system.</p> <p>Symbol Ids can be either system-specific ids (see further below) or be one of the pre-defined OGR 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.</p> <div style="text-align: center;">     </div> <div style="text-align: center;">     </div> <div style="text-align: center;">    </div> <p>Here is the current list of OGR symbol ids (this could grow over time):</p> <ul style="list-style-type: none"> • 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 diag bar) <p>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.).</p> <p>The following conventions will be used for common system-specific symbol ids:</p> <ul style="list-style-type: none"> • "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. <p>Other conventions may be added in the future (such as vector symbols, WMF, etc).</p>
6.7. Feature Style Specification	<p>Angle - Rotation angle (in degrees, counterclockwise) to apply to the symbol.</p>
c	<p>Symbol Color, expressed in hexadecimal</p>

6.7.2.6 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:

param	Description
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	 <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: <i>baseline</i>, <i>center</i>, <i>top</i> and <i>bottom</i>; and three horizontal modes: <i>left</i>, <i>center</i> and <i>right</i>. The scheme is shown at 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.</p>
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). If the offset is negative then the label will be shifted left of the main segment, and right otherwise.
6.7. Feature Style Specification	
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.

6.7.2.7 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)
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.

6.7.2.8 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.

6.7.3 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);

// 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++)
```

(continues on next page)

(continued from previous page)

```

{
    poStyleTool = GetPart(iPart);
    switch (poStyleTool->GetType())
    {
        case OGRSTCPen:
            poStylePen = (OGRStylePen *)poStyleTool;
            pszColor = poStylePen->Color(bDefault);
            if (bDefault == FALSE)
                poStylePen->GetRGBFromString(pszColor, nRed, nGreen,
                                             nBlue, nTrans);

            else
                // Color not defined

            dfWidth = poStylePen->Width(bDefault);
            if (bDefault == FALSE)
                // Use dfWidth
            else
                // dfWidth not defined

            :
            :
    }
}

```

6.7.4 REVISION HISTORY

- **Version 0.016 - 2018-12-03 - Andrew Sudorgin** Restored and documented font property for point symbols
- **Version 0.015 - 2018-01-08 - Alan Thomas** Update outdated material; minor changes to BRUSH 'id' and LABEL 't', 'bo', 'it', 'un', 'st'; clarify BRUSH 'fc', 'bc', SYMBOL 'o' and LABEL 's', 'w', 'p'
- **Version 0.014 - 2011-07-24 - Even Rouault** Mention the escaping of double-quote characters in the text string of a LABEL (ticket #3675)
- **Version 0.013 - 2008-07-29 - Daniel Morissette** Added 'o:' for font point symbol outline color (ticket #2509)
- **Version 0.012 - 2008-07-21 - Daniel Morissette** Added 'o:' for text outline color and updated 'b:' to be specifically a filled label background box (ticket #2480)
- **Version 0.011 - 2008-02-28 - Tamas Szekeres** Note about OGR SQL to transfer the style between the data sources
- **Version 0.010 - 2006-09-23- Andrey Kiselev** Added label styles 'w', 'st', 'h', 'm:h', 'm:a', 'p:{10,11,12}'
- **Version 0.009 - 2005-03-11- Frank Warmerdam** Remove reference to OGRWin, move into ogr distribution
- **Version 0.008 - 2001-03-21- Frank Warmerdam** Fix minor typos (h:12pt instead of s:12pt in examples)
- **Version 0.008 - 2000-07-15 - Stephane Villeneuve** Remove style table in Layer. Add forecolor and backcolor to brush.
- **Version 0.007 - 2000-06-22 - Daniel Morissette** Fixed typo and added offset param for PEN.
- **Version 0.006 - 2000-06-20 - Daniel Morissette** Added the OGR-Win idea and made small changes here and there.
- **Version 0.005 - 2000-06-12 - Daniel Morissette** Allow passing of comma-delimited list of names in PEN's "id" parameter. Defined system-independent pen style names.

- **Version 0.004 - 2000-06-09 - Stephane Villeneuve** Added PEN cap and join parameters More clearly defined the API
- **Version 0.003 - 2000-02-15 - Daniel Morissette** First kind-of-complete version.

6.8 Configuration options

This page discussed runtime configuration options for GDAL, and is 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.

6.8.1 How to set configuration options ?

One example of a config 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
```

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.

6.8.2 List of configuration options and where they apply

Note: This list is known to be incomplete. It depends on proper annotation of configuration options where they are mentioned elsewhere in the documentation. If you want to help to extend it, use the `:decl_configoption: `NAME`` syntax in places where a configuration option is mentioned.

- **AWS_CONFIG_FILE:** *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=YES:** *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_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_VIRTUAL_HOSTING:** *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, ...*
- **COMPRESS_OVERVIEW:** *gdaladdo*
- **CPL_AWS_CREDENTIALS_FILE:** *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- **CPL_GS_CREDENTIALS_FILE:** *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- **CPL_MACHINE_IS_GCE:** *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- **CPL_VSIL_CURL_CACHE_SIZE:** *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- **CPL_VSIL_CURL_CHUNK_SIZE:** *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_NON_CACHED:** *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- **CPL_VSIL_CURL_USE_HEAD:** *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*

- **CPL_VSIL_CURL_USE_S3_REDIRECT**: *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, ...*
- **CURL_CA_BUNDLE**: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- **ESRI_XML_PAM**: *GTiff – GeoTIFF File Format*
- **GDAL_ENABLE_TIFF_SPLIT**: *GTiff – GeoTIFF File Format*
- **GDAL_GEOREF_SOURCES**: *GTiff – GeoTIFF File Format*
- **GDAL_HTTP_HEADER_FILE**: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- **GDAL_HTTP_MAX_RETRY**: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- **GDAL_HTTP_PROXY**: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- **GDAL_HTTP_RETRY_DELAY**: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- **GDAL_NUM_THREADS**:
 - *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
 - *GTiff – GeoTIFF File Format*
- **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**:
 - *VICAR – VICAR*
 - *PDS – Planetary Data System v3*
- **GOOGLE_APPLICATION_CREDENTIALS**: *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_REFRESH_TOKEN**: *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, ...*
- **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_REPORT_COMPD_CS**: *GTiff – GeoTIFF File Format*
- **GTIFF_VIRTUAL_MEM_IO**: *GTiff – GeoTIFF File Format*

- GTIFF_WRITE_TOWGS84: *GTiff – GeoTIFF File Format*
- INTERLEAVE_OVERVIEW: *gdaladdo*
- JPEG_QUALITY_OVERVIEW: *GTiff – GeoTIFF File Format*
- MITAB_SET_TOWGS84_ON_KNOWN_DATUM: *MapInfo TAB and MIF/MID*
- OGR_PDF_READ_NON_STRUCTURED: *PDF – Geospatial PDF*
- OGR_SQL_LIKE_AS_ILIKE: *OGR SQL dialect*
- OSS_SECRET_ACCESS_KEY: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- PHOTOMETRIC_OVERVIEW: *gdaladdo*
- SHAPE_ENCODING: *ESRI Shapefile / DBF*
- SWIFT_AUTH_V1_URL: *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, ...*
- TAB_APPROX_GEOTRANSFORM: *GTiff – GeoTIFF File Format*
- TIFF_USE_OVR: *GTiff – GeoTIFF File Format*
- USE_RRD:
 - *gdaladdo*
 - *GTiff – GeoTIFF File 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: *GDAL Virtual File Systems (compressed, network hosted, etc...): /vsimem, /vsizip, /vsitar, /vsicurl, ...*
- 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, ...*

API

API is omitted in this PDF document. You can consult it on <https://gdal.org/api/index.html>

TUTORIALS

8.1 Raster

8.1.1 Raster API tutorial

8.1.1.1 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*).

In C++:

```
#include "gdal_priv.h"
#include "cpl_conv.h" // for CPLMalloc()
int main()
{
    GDALDataset *poDataset;
    GDALAllRegister();
    poDataset = (GDALDataset *) GDALOpen( pszFilename, GA_ReadOnly );
    if( poDataset == NULL )
    {
        ...;
    }
}
```

In C:

```
#include "gdal.h"
#include "cpl_conv.h" /* for CPLMalloc() */
int main()
{
    GDALDatasetH hDataset;
    GDALAllRegister();
    hDataset = GDALOpen( pszFilename, GA_ReadOnly );
    if( hDataset == NULL )
    {
        ...;
    }
}
```

In Python:

```
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.

8.1.1.2 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 takes the following form :

```
adfGeoTransform[0] /* top left x */
adfGeoTransform[1] /* w-e pixel resolution */
adfGeoTransform[2] /* 0 */
adfGeoTransform[3] /* top left y */
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:

In C++:

```
double          adfGeoTransform[6];
printf( "Driver: %s/%s\n",
        poDataset->GetDriver()->GetDescription(),
        poDataset->GetDriver()->GetMetadataItem( GDAL_DMD_LONGNAME ) );
printf( "Size is %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] );
}
```

In C:

```
GDALDriverH      hDriver;
double           adfGeoTransform[6];
```

(continues on next page)

(continued from previous page)

```

hDriver = GDALGetDatasetDriver( hDataset );
printf( "Driver: %s/%s\n",
        GDALGetDriverShortName( hDriver ),
        GDALGetDriverLongName( hDriver ) );
printf( "Size is %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] );
}

```

In Python:

```

print("Driver: {}".format(dataset.GetDriver().ShortName,
                          dataset.GetDriver().LongName))
print("Size is {} x {} x {}".format(dataset.RasterXSize,
                                     dataset.RasterYSize,
                                     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]))

```

8.1.1.3 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 `GDALRasterBand::GetRasterCount()`) and displays a little information about it.

In C++:

```

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 );

```

(continues on next page)

(continued from previous page)

```
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() );
```

In C:

```
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 ) ) );
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 ) ) );
```

In Python:

```
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()))
```

8.1.1.4 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.

In C++:

```
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 );
```

The `pafScanline` buffer should be freed with `CPLFree()` when it is no longer used.

In C:

```
float *pafScanline;
int  nXSize = GDALGetRasterBandXSize( hBand );
pafScanline = (float *) CPLMalloc(sizeof(float)*nXSize);
GDALRasterIO( hBand, GF_Read, 0, 0, nXSize, 1,
               pafScanline, nXSize, 1, GDT_Float32,
               0, 0 );
```

The `pafScanline` buffer should be freed with `CPLFree()` when it is no longer used.

In Python:

```
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 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_Byte`. 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_Byte` 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.

8.1.1.5 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.

8.1.1.6 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()*.

In C++:

```
#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 );
```

In C:

```
#include "cpl_string.h"
...
```

(continues on next page)

(continued from previous page)

```

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 );

```

In Python:

```

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().

8.1.1.7 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:

In C++:

```

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 );

```

In C:

```

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);

```

In Python:

```

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:

In C++:

```

#include "cpl_string.h"
...
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 );

```

In C:

```

#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 );

```

In Python:

```

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

```

8.1.1.8 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.

In C++:

```
GDALDataset *poDstDS;
char **papszOptions = NULL;
poDstDS = poDriver->Create( pszDstFilename, 512, 512, 1, GDT_Byte,
                           papszOptions );
```

In C:

```
GDALDatasetH hDstDS;
char **papszOptions = NULL;
hDstDS = GDALCreate( hDriver, pszDstFilename, 512, 512, 1, GDT_Byte,
                    papszOptions );
```

In Python:

```
dst_ds = driver.Create(dst_filename, xsize=512, ysize=512,
                      bands=1, eType=gdal.GDT_Byte)
```

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.

In C++:

```
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_Byte, 0, 0 );
/* Once we're done, close properly the dataset */
GDALClose( (GDALDatasetH) poDstDS );
```

In C:

```
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 );
```

(continues on next page)

(continued from previous page)

```

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_Byte, 0, 0 );
/* Once we're done, close properly the dataset */
GDALClose( hDstDS );

```

In Python:

```

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")
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

```

8.1.2 Raster driver implementation tutorial

8.1.2.1 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.

8.1.2.2 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 : public GDALPamDataset
{
    friend class JDEMRasterBand;
    FILE          *fp;
    GByte          abyHeader[1012];

public:
    ~JDEMDataset();
    static GDALDataset *Open( GDALOpenInfo * );
    static int          Identify( GDALOpenInfo * );

```

(continues on next page)

(continued from previous page)

```

CPLERR      GetGeoTransform( double * padfTransform );
const char *GetProjectionRef();
};

```

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.
    if( !Identify(poOpenInfo) )
        return NULL;

    // 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 NULL;
    }

    // Check that the file pointer from GDALOpenInfo* is available
    if( poOpenInfo->fpL == NULL )
    {
        return NULL;
    }

    // Create a corresponding GDALDataset.
    JDEMDataset *poDS = new JDEMDataset();

    // Borrow the file pointer from GDALOpenInfo*.
    poDS->fp = poOpenInfo->fpL;
    poOpenInfo->fpL = NULL;

    // Read the header.
    VSIFReadL(poDS->abyHeader, 1, 1012, poDS->fp);
    poDS->nRasterXSize =
        JDEMGetField(reinterpret_cast<char *>(poDS->abyHeader) + 23, 3);
    poDS->nRasterYSize =
        JDEMGetField(reinterpret_cast<char *>(poDS->abyHeader) + 26, 3);
    if( poDS->nRasterXSize <= 0 || poDS->nRasterYSize <= 0 )
    {
        CPLError(CE_Failure, CPLE_AppDefined,
            "Invalid dimensions : %d x %d",
            poDS->nRasterXSize, poDS->nRasterYSize);
        delete poDS;
        return NULL;
    }

    // Create band information objects.
    poDS->SetBand(1, new JDEMRasterBand(poDS, 1));
}

```

(continues on next page)

(continued from previous page)

```

// Initialize any PAM information.
poDS->SetDescription(poOpenInfo->pszFilename);
poDS->TryLoadXML();

// Initialize default overviews.
poDS->oOvManager.Initialize(poDS, poOpenInfo->pszFilename);
return poDS;
}

```

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 NULL 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 )
{
    // Confirm that the header has what appears to be dates in the
    // expected locations.  Sadly this is a relatively weak test.
    if( poOpenInfo->nHeaderBytes < 50 )
        return FALSE;

    // Check if century values seem reasonable.
    const char *psHeader = reinterpret_cast<char *>(poOpenInfo->pabyHeader);
    if( (!EQUALN(psHeader + 11, "19", 2) &&
        !EQUALN(psHeader + 11, "20", 2)) ||
        (!EQUALN(psHeader + 15, "19", 2) &&
        !EQUALN(psHeader + 15, "20", 2)) ||
        (!EQUALN(psHeader + 19, "19", 2) &&
        !EQUALN(psHeader + 19, "20", 2)) )
    {
        return FALSE;
    }
}

```

(continues on next page)

(continued from previous page)

```

    }
    return TRUE;
}

```

It is important to make the “is this my format” test as stringent as possible. In this particular case the test is weak, and a file that happened to have 19s or 20s at a few locations could be erroneously recognized as JDEM format, causing it to not be handled properly. 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.

```

// Check that the file pointer from GDALOpenInfo* is available.
if( poOpenInfo->fpL == NULL )
{
    return NULL;
}
JDEMDataset *poDS = new JDEMDataset();

// Borrow the file pointer from GDALOpenInfo*.
poDS->fp = poOpenInfo->fpL;
poOpenInfo->fpL = NULL;

```

At this point we “borrow” the file handle that was held by GDALOpenInfo*. 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.

```

VSIFReadL(poDS->abyHeader, 1, 1012, poDS->fp);
poDS->nRasterXSize =
    JDEMGetField(reinterpret_cast<char *>(poDS->abyHeader) + 23, 3);
poDS->nRasterYSize =
    JDEMGetField(reinterpret_cast<char *>(poDS->abyHeader) + 26, 3);
if (poDS->nRasterXSize <= 0 || poDS->nRasterYSize <= 0 )
{
    CPLError(CE_Failure, CPLE_AppDefined,
        "Invalid dimensions : %d x %d",
        poDS->nRasterXSize, poDS->nRasterYSize);
    delete poDS;
    return NULL;
}

```

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, 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. For more details on these services review the GDAL-PamDataset and related classes.

```
// Initialize any PAM information.
poDS->SetDescription( poOpenInfo->pszFilename );
poDS->TryLoadXML();
return poDS;
}
```

8.1.2.3 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 : public GDALPamRasterBand
{
public:
    JDEMRasterBand( JDEMDataSet *, int );
    virtual CPLErr IReadBlock( int, int, void * );
};
```

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 )
{
    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_Byte, 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 = static_cast<JDEMDataSet *>(poDS);
```

(continues on next page)

(continued from previous page)

```

int nRecordSize = nBlockXSize * 5 + 9 + 2;
VSIFSeekL(poGDS->fp, 1011 + nRecordSize*nBlockYOff, SEEK_SET);
char *pszRecord = static_cast<char *>(CPLMalloc(nRecordSize));
VSIFReadL(pszRecord, 1, nRecordSize, poGDS->fp);
if( !EQUALN(reinterpret_cast<char *>(poGDS->abyHeader), pszRecord, 6) )
{
    CPLFree(pszRecord);
    CPLError(CE_Failure, CPLE_AppDefined,
        "JDEM Scanline corrupt. Perhaps file was not transferred "
        "in binary mode?");
    return CE_Failure;
}
if( JDEMGetField(pszRecord + 6, 3) != nBlockYOff + 1 )
{
    CPLFree(pszRecord);
    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++ )
    ((float *) pImage)[i] = JDEMGetField(pszRecord + 9 + 5 * i, 5) * 0.1;
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.

8.1.2.4 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") != NULL )
        return;
}

```

(continues on next page)

(continued from previous page)

```

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,
    "frmt_various.html#JDEM");
poDriver->SetMetadataItem(GDAL_DMD_EXTENSION, "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 (starting with GDAL 1.5.0). This is an optional macro for drivers inside GDAL tree that don't depend on external libraries, but that can be very useful if you compile your driver as a plugin (that is to say, an out-of-tree driver). As the GDAL C++ ABI may, and will, change between GDAL releases (for example from GDAL 1.5.0 to 1.6.0), it may be necessary to recompile your driver against the header files of the GDAL 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.

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 identify 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_EXTENSION`: The extension used for files of this type. If more than one pick the primary extension, or none at all. (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)

8.1.2.5 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 `gdal/frmts`, with the directory name the same as the short name.
- Add a GNUmakefile and `makefile.vc` in that directory modeled on those from other similar directories (i.e. the `jdem` directory).
- 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 `gdal/gcore/gdal_frmts.h`.
- Add a call to the registration function to `frmts/gdalallregister.cpp`, protected by an appropriate `#ifdef`.
- Add the format short name to the `GDAL_FORMATS` macro in `GDALmake.opt.in` (and to `GDALmake.opt`).
- Add a format specific item to the `EXTRAFLAGS` macro in `frmts/makefile.vc`.

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.

8.1.2.6 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 );
const char *GetProjectionRef();
```

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<char *>(abyHeader);
    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();
}
```

(continues on next page)

(continued from previous page)

```

    padfTransform[2] = 0.0;
    padfTransform[4] = 0.0;
    padfTransform[5] = -1 * (dfURLat - dfLLLat) / GetRasterYSize();
    return CE_None;
}

```

The `GDALDataset::GetProjectionRef()` method returns a pointer to an internal string containing a coordinate system definition in OGC WKT format. In this case the coordinate system is fixed for all files of this format, 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.

```

const char *JDEMDataset::GetProjectionRef()
{
    return
        "GEOGCS[\"Tokyo\", DATUM[\"Tokyo\", SPHEROID[\"Bessel 1841\", \"
        \"6377397.155, 299.1528128, AUTHORITY[\"EPSG\", 7004]], TOWGS84[-148, \"
        \"507, 685, 0, 0, 0, 0], AUTHORITY[\"EPSG\", 6301]], PRIMEM[\"Greenwich\", \"
        \"0, AUTHORITY[\"EPSG\", 8901]], UNIT[\"DMSH\", 0.0174532925199433, \"
        \"AUTHORITY[\"EPSG\", 9108]], AXIS[\"Lat\", NORTH], AXIS[\"Long\", EAST], \"
        \"AUTHORITY[\"EPSG\", 4301]]\";
}

```

This completes explanation of the features of the JDEM driver. The full source for `jdemdataset.cpp` can be reviewed as needed.

8.1.2.7 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/geotiff/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, 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.

8.1.2.8 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 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 rudimentary checks
    if( nBands != 1 && nBands != 3 )
    {
        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_Byte && bStrict )
```

(continues on next page)

(continued from previous page)

```

{
    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);
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);

```

(continues on next page)

(continued from previous page)

```

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_Byte,
                            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,
                                  int nXSize, int nYSize, int nBands,
                                  GDALDataType eType,
                                  char ** /* papszParmList */ )
{
    // Verify input options.
    if( eType != GDT_Byte && 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;
    }
}

```

(continues on next page)

(continued from previous page)

```

// 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;
}

// 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, "AuxiliaryTarget: %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 = GDALGetDataTypeInfo(eType)/8;

```

(continues on next page)

(continued from previous page)

```

    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.

8.1.2.9 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 format, 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_Byte, TRUE));
}
else
{
    poDS->SetBand(
        1, new RawRasterBand(poDS, 1, poDS->fpImage,
                             iIn, 3, nWidth*3, GDT_Byte, TRUE));
    poDS->SetBand(

```

(continues on next page)

(continued from previous page)

```

        2, new RawRasterBand(poDS, 2, poDS->fpImage,
                           iIn+1, 3, nWidth*3, GDT_Byte, TRUE));
    poDS->SetBand(
        3, new RawRasterBand(poDS, 3, poDS->fpImage,
                           iIn+2, 3, nWidth*3, GDT_Byte, 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.

8.1.2.10 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_Byte 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.

8.1.3 GDAL Warp API tutorial (Reprojection, ...)

8.1.3.1 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.

8.1.3.2 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 =
        (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.
```

(continues on next page)

(continued from previous page)

```

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.

8.1.3.3 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.

8.1.3.4 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 );

// 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.
```

(continues on next page)

(continued from previous page)

```

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.

8.1.3.5 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 since GDAL 1.8.0, for previous versions -with-threads was required in configure).
- The resampling kernels vary in work required from nearest neighbour being least, then bilinear then cubic. Don't use a more complex resampling kernel than needed.
- Avoid use of esoteric masking options so that special simplified logic can 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.

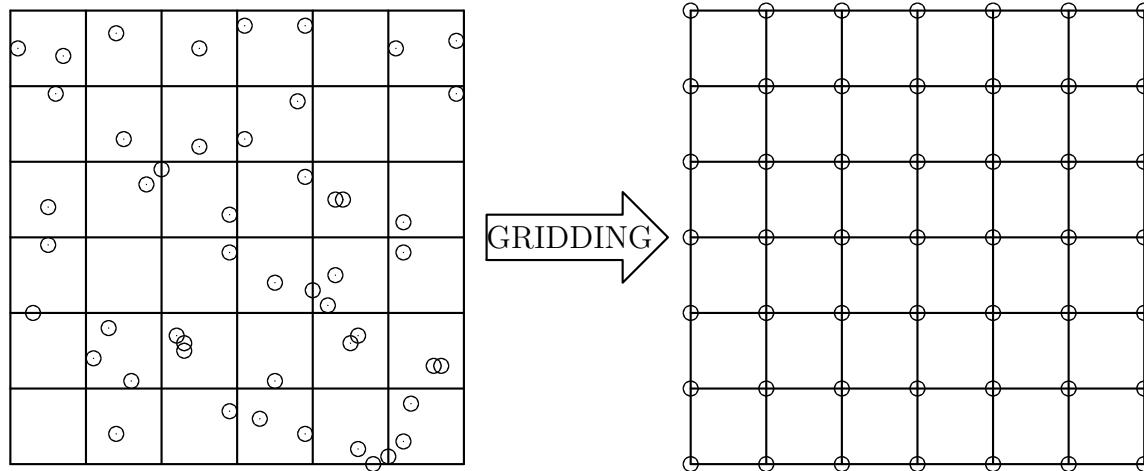
8.1.3.6 Other Masking Options

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.

8.1.4 GDAL Grid Tutorial

8.1.4.1 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 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 is valuable itself 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. The rest of this document discusses details on algorithms and their parameters implemented in GDAL Grid API.

8.1.4.2 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 interpolated value for the given position in output grid.

For every grid node the resulting value Z will be calculated using 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 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 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.

8.1.4.3 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.

$$Z = \min(Z_1, Z_2, \dots, Z_n)$$

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 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.

$$Z = \max(Z_1, Z_2, \dots, Z_n)$$

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 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.

$$Z = \max(Z_1, Z_2, \dots, Z_n) - \min(Z_1, Z_2, \dots, Z_n)$$

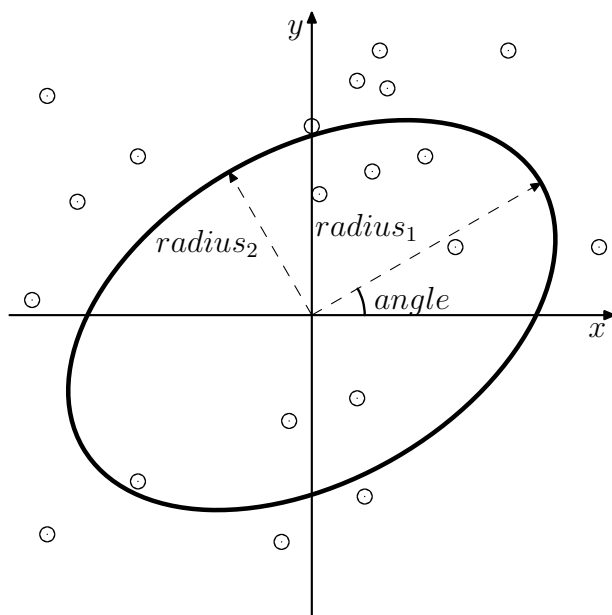
where:

- 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 Ellipse*.

8.1.4.4 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.

8.2 Multidimensional raster

8.2.1 Multidimensional raster API tutorial

8.2.1.1 Read the content of an array

In C++

```
#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 )
    {
        exit(1);
    }
    size_t nValues = 1;
    std::vector<size_t> anCount;
```

(continues on next page)

(continued from previous page)

```

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;
}

```

In C

```

#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 )
    {
        exit(1);
    }

    dims = GDALMDArrayGetDimensions(hVar, &nDimCount);
    panCount = (size_t*)CPLMalloc(nDimCount * sizeof(size_t));

```

(continues on next page)

(continued from previous page)

```

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;
}

```

In Python

```

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:

```

from osgeo import gdal
ds = gdal.OpenEx("in.nc", gdal.OF_MULTIDIM_RASTER)
rootGroup = ds.GetRootGroup()
var = rootGroup.OpenMDArray("temperature")
data = var.ReadAsArray(buffer_datatype = gdal.ExtendedDataType.Create(gdal.GDT_
↪Float64))

```

8.2.1.2 Other examples

Test scripts from the GDAL autotest suite

- <https://raw.githubusercontent.com/OSGeo/gdal/master/autotest/gdrivers/memmultidim.py>
- https://raw.githubusercontent.com/OSGeo/gdal/master/autotest/gdrivers/netcdf_multidim.py
- <https://raw.githubusercontent.com/OSGeo/gdal/master/autotest/gdrivers/vrtmultidim.py>
- https://raw.githubusercontent.com/OSGeo/gdal/master/autotest/utilities/test_gdalmdiminfo_lib.py
- https://raw.githubusercontent.com/OSGeo/gdal/master/autotest/utilities/test_gdalmdimtranslate_lib.py

8.3 Vector

8.3.1 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 read 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.

8.3.1.1 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.

In C++ :

```
#include "ogr_sfrmts.h"

int main()
{
    GDALAllRegister();
}
```

In C :

```
#include "gdal.h"

int main()
{
    GDALAllRegister();
}
```

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.

In C++ :

```
GDALDataset      *poDS;

poDS = (GDALDataset*) GDALOpenEx( "point.shp", GDAL_OF_VECTOR, NULL, NULL, NULL );
if( poDS == NULL )
{
    printf( "Open failed.\n" );
    exit( 1 );
}
```

In C :

```
GDALDatasetH hDS;

hDS = GDALOpenEx( "point.shp", GDAL_OF_VECTOR, NULL, NULL, NULL );
if( hDS == NULL )
{
    printf( "Open failed.\n" );
    exit( 1 );
}
```

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.

In C++ :

```
OGRLayer *poLayer;

poLayer = poDS->GetLayerByName( "point" );
```

In C :

```
OGRLayerH hLayer;

hLayer = GDALDatasetGetLayerByName( hDS, "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.

With GDAL 2.3 and C++11:

```
for( auto& poFeature: poLayer )
{
```

With GDAL 2.3 and C:

```
OGR_FOR_EACH_FEATURE_BEGIN(hFeature, hLayer)
{
```

If using older GDAL versions, while it isn't strictly necessary in this circumstance since we are starting fresh with the layer, it is often wise to call `OGRLayer::ResetReading()` to ensure we are starting at the beginning of the layer. We iterate through all the features in the layer using `OGRLayer::GetNextFeature()`. It will return `NULL` when we run out of features.

With GDAL > 2.3 and C++ :

```
OGRFeature *poFeature;
```

(continues on next page)

(continued from previous page)

```
poLayer->ResetReading();
while( (poFeature = poLayer->GetNextFeature()) != NULL )
{
```

With GDAL > 2.3 and C :

```
OGRFeatureH hFeature;

OGR_L_ResetReading(hLayer);
while( (hFeature = OGR_L_GetNextFeature(hLayer)) != NULL )
{
```

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.

With GDAL 2.3 and C++11:

```
for( auto&& oField: *poFeature )
{
    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:
            printf( "%s, ", oField.GetString() );
            break;
        default:
            printf( "%s, ", oField.GetAsString() );
            break;
    }
}
```

With GDAL > 2.3 and C++ :

```
OGRFeatureDefn *poFDefn = poLayer->GetLayerDefn();
for( int iField = 0; iField < poFDefn->GetFieldCount(); iField++ )
{
    OGRFieldDefn *poFieldDefn = poFDefn->GetFieldDefn( iField );

    switch( poFieldDefn->GetType() )
    {
        case OFTInteger:
            printf( "%d, ", poFeature->GetFieldAsInteger( iField ) );
            break;
        case OFTInteger64:
            printf( CPL_FRMT_GIB " ", poFeature->GetFieldAsInteger64( iField ) );
            break;
        case OFTReal:
            printf( "%.3f, ", poFeature->GetFieldAsDouble( iField ) );
```

(continues on next page)

(continued from previous page)

```

        break;
    case OFTString:
        printf( "%s", poFeature->GetFieldAsString(iField) );
        break;
    default:
        printf( "%s", poFeature->GetFieldAsString(iField) );
        break;
    }
}

```

In C :

```

OGRFeatureDefnH hFDefn = OGR_L_GetLayerDefn(hLayer);
int iField;

for( iField = 0; iField < OGR_FD_GetFieldCount(hFDefn); iField++ )
{
    OGRFieldDefnH hFieldDefn = OGR_FD_GetFieldDefn( hFDefn, iField );

    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;
    }
}

```

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.

In C++ :

```

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();
    
```

(continues on next page)

(continued from previous page)

```

#else
    OGRPoint *poPoint = (OGRPoint *) poGeometry;
#endif

    printf( "%.3f,%.3f\n", poPoint->getX(), poPoint->getY() );
}
else
{
    printf( "no point geometry\n" );
}

```

In C :

```

OGRGeometryH hGeometry;

hGeometry = OGR_F_GetGeometryRef(hFeature);
if( hGeometry != NULL
    && wkbFlatten(OGR_G_GetGeometryType(hGeometry)) == wkbPoint )
{
    printf( "%.3f,%.3f\n", OGR_G_GetX(hGeometry, 0), OGR_G_GetY(hGeometry, 0) );
}
else
{
    printf( "no point geometry\n" );
}

```

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.

Starting with OGR 1.11, several geometry fields can be associated to a feature.

In C++ :

```

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,%.3f\n", poPoint->getX(), poPoint->getY() );
    }
    else
    {
        printf( "no point geometry\n" );
    }
}

```

(continues on next page)

(continued from previous page)

}

In C :

```

OGRGeometryH hGeometry;
int iGeomField;
int nGeomFieldCount;

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,%.3f\n", OGR_G_GetX(hGeometry, 0),
                OGR_G_GetY(hGeometry, 0) );
    }
    else
    {
        printf( "no point geometry\n" );
    }
}

```

In Python:

```

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 deleted the return geometry.

With GDAL 2.3 and C++11, the looping over features is simply terminated by a closing curly bracket.

}

With GDAL 2.3 and C, the looping over features is simply terminated by the following.

```

}
OGR_FOR_EACH_FEATURE_END(hFeature)

```

For GDAL > 2.3, as the `OGRLayer::GetNextFeature()` method returns a copy of the feature that is now owned by us. So at the end of use we must free the feature. We could just “delete” it, but this can cause problems in windows builds where the GDAL DLL has a different “heap” from the main program. To be on the safe side we use a GDAL function to delete the feature.

In C++ :

```

OGRFeature::DestroyFeature( poFeature );
}

```

In C :

```
OGR_F_Destroy( hFeature );
}
```

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.

In C/C++ :

```
GDALClose( poDS );
}
```

All together our program looks like this.

With GDAL 2.3 and C++11 :

```
#include "ogr_sfrmts.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 )
            {
                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:
                        printf( "%s", oField.GetString() );
                        break;
                    default:
                        printf( "%s", oField.GetAsString() );
                        break;
                }
            }

            const OGRGeometry *poGeometry = poFeature->GetGeometryRef();
            if( poGeometry != nullptr
```

(continues on next page)

(continued from previous page)

```

        && 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;
}

```

In C++ :

```

#include "ogr_sfrmts.h"

int main()
{
    GDALAllRegister();

    GDALDataset *poDS = static_cast<GDALDataset*>(
        GDALOpenEx( "point.shp", GDAL_OF_VECTOR, NULL, NULL, NULL ));
    if( poDS == NULL )
    {
        printf( "Open failed.\n" );
        exit( 1 );
    }

    OGRLayer *poLayer = poDS->GetLayerByName( "point" );
    OGRFeatureDefn *poFDefn = poLayer->GetLayerDefn();

    poLayer->ResetReading();
    OGRFeature *poFeature;
    while( (poFeature = poLayer->GetNextFeature()) != NULL )
    {
        for( int iField = 0; iField < poFDefn->GetFieldCount(); iField++ )
        {
            OGRFieldDefn *poFieldDefn = poFDefn->GetFieldDefn( iField );

            switch( poFieldDefn->GetType() )
            {
                case OFTInteger:
                    printf( "%d,", poFeature->GetFieldAsInteger( iField ) );
                    break;
                case OFTInteger64:
                    printf( CPL_FRMT_GIB " ", poFeature->GetFieldAsInteger64( iField ) );
                    break;
                case OFTReal:
                    printf( "%.3f,", poFeature->GetFieldAsDouble( iField ) );
                    break;
                case OFTString:
                    printf( "%s,", poFeature->GetFieldAsString( iField ) );
                    break;
            }
        }
        printf( "\n" );
    }
}

```

(continues on next page)

(continued from previous page)

```

        break;
    default:
        printf( "%s", poFeature->GetFieldAsString(iField) );
        break;
    }
}

OGRGeometry *poGeometry = poFeature->GetGeometryRef();
if( poGeometry != NULL
    && wkbFlatten(poGeometry->getGeometryType()) == wkbPoint )
{
    OGRPoint *poPoint = (OGRPoint *) poGeometry;

    printf( "%.3f,%.3f\n", poPoint->getX(), poPoint->getY() );
}
else
{
    printf( "no point geometry\n" );
}
OGRFeature::DestroyFeature( poFeature );
}

GDALClose( poDS );
}

```

In C :

```

#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++ )
        {

```

(continues on next page)

(continued from previous page)

```

OGRFieldDefnH hFieldDefn = OGR_FD_GetFieldDefn( hFDefn, iField );

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 );
}

```

In Python:

```

import sys
from osgeo import gdal

ds = gdal.OpenEx( "point.shp", gdal.OF_VECTOR )
if ds is None:
    print "Open failed.\n"
    sys.exit( 1 )

lyr = ds.GetLayerByName( "point" )

lyr.ResetReading()

for feat in lyr:

```

(continues on next page)

(continued from previous page)

```

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\n"

ds = None

```

8.3.1.2 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.

In C++ :

```

#include "ogr_srf_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 );
    }
}

```

In C :

```

#include "ogr_api.h"

int main()
{

```

(continues on next page)

(continued from previous page)

```

const char *pszDriverName = "ESRI Shapefile";
GDALDriver *poDriver;

GDALAllRegister();

poDriver = (GDALDriver*) GDALGetDriverByName(pszDriverName );
if( poDriver == NULL )
{
    printf( "%s driver not available.\n", pszDriverName );
    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.

In C++:

```

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 );
}

```

In C:

```

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 );
}

```

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.

In C++:

```

OGRLayer *poLayer;

poLayer = poDS->CreateLayer( "point_out", NULL, wkbPoint, NULL );
if( poLayer == NULL )
{
    printf( "Layer creation failed.\n" );
    exit( 1 );
}

```

In C:

```
OGRLayerH hLayer;

hLayer = GDALDatasetCreateLayer( hDS, "point_out", NULL, wkbPoint, NULL );
if( hLayer == NULL )
{
    printf( "Layer creation failed.\n" );
    exit( 1 );
}
```

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.

In C++:

```
OGRFieldDefn oField( "Name", OFTString );

oField.SetWidth(32);

if( poLayer->CreateField( &oField ) != OGRERR_NONE )
{
    printf( "Creating Name field failed.\n" );
    exit( 1 );
}
```

In C:

```
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);
```

The following snippets loop reading lines of the form “x,y,name” from stdin, and parsing them.

In C++ and in C :

```
double x, y;
char szName[33];

while( !feof(stdin)
    && fscanf( stdin, "%lf,%lf,%32s", &x, &y, szName ) == 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.

In C++ :

```
OGRFeature *poFeature;

poFeature = OGRFeature::CreateFeature( poLayer->GetLayerDefn() );
poFeature->SetField( "Name", szName );
```

In C :

```
OGRFeatureH hFeature;

hFeature = OGR_F_Create( OGR_L_GetLayerDefn( hLayer ) );
OGR_F_SetFieldString( hFeature, OGR_F_GetFieldIndex(hFeature, "Name"), szName );
```

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.

In C++ :

```
OGRPoint pt;
pt.setX( x );
pt.setY( y );

poFeature->SetGeometry( &pt );
```

In C :

```
OGRGeometryH hPt;
hPt = OGR_G_CreateGeometry(wkbPoint);
OGR_G_SetPoint_2D(hPt, 0, x, y);

OGR_F_SetGeometry( hFeature, hPt );
OGR_G_DestroyGeometry(hPt);
```

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.

In C++ :

```
if( poLayer->CreateFeature( poFeature ) != OGRERR_NONE )
{
    printf( "Failed to create feature in shapefile.\n" );
    exit( 1 );
}

OGRFeature::DestroyFeature( poFeature );
}
```

In C :

```
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 );
}

```

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.

In C/C++ :

```

GDALClose( poDS );
}

```

The same program all in one block looks like this:

In C++ :

```

#include "ogr_sfrmts.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);

    if( poLayer->CreateField( &oField ) != OGRERR_NONE )
    {

```

(continues on next page)

(continued from previous page)

```

    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 );
}

```

In C :

```

#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 )
    {
        printf( "%s driver not available.\n", pszDriverName );
        exit( 1 );
    }
}

```

(continues on next page)

(continued from previous page)

```

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 );
    }

    OGR_F_Destroy( hFeature );
}

GDALClose( hDS );
}

```

In Python :

```

import sys
from osgeo import gdal
from osgeo import ogr
import string

driverName = "ESRI Shapefile"
drv = gdal.GetDriverByName( driverName )
if drv is None:
    print "%s driver not available.\n" % driverName
    sys.exit( 1 )

ds = drv.Create( "point_out.shp", 0, 0, 0, gdal.GDT_Unknown )
if ds is None:
    print "Creation of output file failed.\n"
    sys.exit( 1 )

lyr = ds.CreateLayer( "point_out", None, ogr.wkbPoint )
if lyr is None:
    print "Layer creation failed.\n"
    sys.exit( 1 )

field_defn = ogr.FieldDefn( "Name", ogr.OFTString )
field_defn.SetWidth( 32 )

if lyr.CreateField ( field_defn ) != 0:
    print "Creating Name field failed.\n"
    sys.exit( 1 )

# Expected format of user input: x y name
linestring = raw_input()
linelist = string.split(linestring)

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)

    if lyr.CreateFeature(feat) != 0:
        print "Failed to create feature in shapefile.\n"
        sys.exit( 1 )

    feat.Destroy()

    linestring = raw_input()
    linelist = string.split(linestring)

ds = None

```

Starting with OGR 1.11, 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.

In 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 oPointField2( "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 );
}
```

In C :

```
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);

if( OGR_L_CreateGeomField( hLayer, hPointField2 ) != OGRERR_NONE )
{
    printf( "Creating field PointField2 failed.\n" );
    exit( 1 );
}
```

(continues on next page)

(continued from previous page)

```

}

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.

In C++ :

```

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 );

```

In C :

```

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 );
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 );

```

In Python :

```
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 );
}
```

8.3.2 Vector driver implementation tutorial

8.3.2.1 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.

8.3.2.2 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.

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.

8.3.2.3 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; }
    OGRSPFLayer      *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;
    papoLayers = static_cast<OGRSPFLayer **>(CPLMalloc(sizeof(void *)));
```

(continues on next page)

(continued from previous page)

```

    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];
}

```

8.3.2.4 Read Only Layer

The `OGRSPFLayer` 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 )
{
    nNextFID = 0;

    poFeatureDefn = new OGRFeatureDefn( CPLGetBasename( pszFilename ) );
}

```

(continues on next page)

(continued from previous page)

```

    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);

        pszLine = strstr(pszLine, "|");

        const char *pszName = NULL;
        if( pszLine == NULL )
            continue; // we should issue an error!
        else

```

(continues on next page)

(continued from previous page)

```
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.

8.3.3 Vector driver in Python implementation tutorial

New in version 3.1.

8.3.3.1 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.

8.3.3.2 Linking mechanism to a Python interpreter

See *Linking mechanism to a Python interpreter*

8.3.3.3 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.

8.3.3.4 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.

8.3.3.5 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 = "some_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 description 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 = "url_to_hep_page"`
- `# gdal: DRIVER_DMD_OPENOPTIONLIST = xml_value` where `xml_value` is an `OptionOptionList` specification, like `"<OpenOptionList><Option name='OPT1' type='boolean' description='bla' default='NO'/></OpenOptionList>"`
- and all other metadata items found in `gdal.h` starting with `GDAL_DMD_` (resp. `GDAL_DCAP_`) by creating an item name which starts with `# gdal: DRIVER_` and the value of the `GDAL_DMD_` (resp. `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 super plugin"
# gdal: DRIVER_DMD_EXTENSIONS = "foo bar"
# gdal: DRIVER_DMD_HELPTOPIC = "http://example.com/my_help.html"
```

8.3.3.6 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** (*binary*) – 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** (*binary*) – 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)
```

8.3.3.7 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.

8.3.3.8 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 be 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 iterator must return a dictionary with the feature content.

Two keys allowed in the returned dictionary are:

id

Strongly recommended. The value must be of type `int` to be recognized as a FID by GDAL

type

Required. The value must be the string “OGRFeature”

fields

Required. The value must be 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 a geometry encoded as WKT, 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 *string* (*cap*) – 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.

8.3.3.9 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,
```

(continues on next page)

(continued from previous page)

```

# 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

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.GetMetadataDict()
        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

```

(continues on next page)

(continued from previous page)

```

    # 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)

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_ogrsgf 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.ExportToIsoWkt()
    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))

```

(continues on next page)

(continued from previous page)

```

        for idx in range(gdal_ds.GetLayerCount()):
            self.metadata = gdal_ds.GetMetadata_Dict()

    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)

```

8.3.3.10 Other examples

Other examples, including a CityJSON driver, may be found at <https://github.com/OSGeo/gdal/tree/master/gdal/examples/pydrivers>

8.4 Geographic Network Model

8.4.1 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.

8.4.1.1 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>

```

(continues on next page)

(continued from previous page)

```

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");

```

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 editings. 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 values 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)
{
    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");

    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() );
    }
}
```

(continues on next page)

(continued from previous page)

```

    }

    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_None)
    {
        printf("Can not connect features\n");
    }

    GDALClose(poDS);
}

```

8.4.1.2 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)
{
    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
    {
        printf("Path saved successfully\n");
    }

    GDALClose(poResDS);
    poNet->ReleaseResultSet(poRout);
    GDALClose(poNet);
}

```

8.5 Projections and Spatial Reference Systems tutorial (OSR - OGRSpatialReference)

8.5.1 OGR Coordinate Reference Systems and Coodinate Transformation tutorial

8.5.1.1 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 ESRI’s 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],
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],
```

(continues on next page)

(continued from previous page)

```

    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 imperically, according to what seemed to work for the test data available.

I am prepared to adhere to the Cadorp 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.4](#) 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](#).

8.5.1.2 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 datum) 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.

8.5.1.3 References and applicable standards

- [PROJ documentation](#): projection methods and coordinate operations
- [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

8.5.1.4 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 an 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,
```

(continues on next page)

(continued from previous page)

```
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.

8.5.1.5 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 fdefault value when instantiating *OGRSpatialReference*.
- *OAMS_CUSTOM* means that the data axis 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.

8.5.1.6 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]],
    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()`.

8.5.1.7 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) );
}
...

```

8.5.1.8 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 recognise 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 `SetDataAxisToSRSAxisMapping(OAMS_TRADITIONAL_GIS_ORDER)`

```
OGRSpatialReference oUTM, *poLongLat;
OGRCoordinateTransformation *poTransform;

oUTM.SetProjCS("UTM 17 / WGS84");
oUTM.SetWellKnownGeogCS( "WGS84" );
oUTM.SetUTM( 17 );

poLongLat = oUTM.CloneGeogCS();
poLongLat->SetDataAxisToSRSAxisMapping(OAMS_TRADITIONAL_GIS_ORDER);

poTransform = OGRCreateCoordinateTransformation( &oUTM, poLongLat );
if( poTransform == NULL )
{
    ...
}

...

if( !poTransform->Transform( nPoints, x, y, z ) )
...

```

8.5.1.9 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 have each 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 those area of use. For example, there are several dozains 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

```

(continues on next page)

(continued from previous page)

```

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(
//     "urn:ogc:def:coordinateOperation:EPSG::8599", false);

poTransform = OGRCreatCoordinateTransformation( &oNAD27, &oWGS84, options );

```

8.5.1.10 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 );

```

(continues on next page)

(continued from previous page)

```

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,
                    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 * pszParmName,
                       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,

```

(continues on next page)

(continued from previous page)

```
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):
```

8.5.1.11 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 was mostly containing a 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.4 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.

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.

9.1 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>

9.2 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.

9.3 Chat

You can find some GDAL developers in the IRC channel *#gdal* on *Freenode* which is bridged to a *#gdal:osgeo.org* Matrix room. 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.

Note: the Freenode channel only accepts messages from registered users, so make sure to complete that step. Instructions can be found on <https://wiki.osgeo.org/wiki/Matrix> for doing it from the Matrix side and <https://freenode.net/kb/answer/registration> for doing it from the IRC side (if you don't use Matrix)

9.4 Conference



FOSS4G 2020 is the leading annual conference for free and open source geospatial software. It will include presentations related to GDAL/OGR, and some of the GDAL/OGR development community will be attending. It is the event for those interested in GDAL/OGR, other FOSS geospatial technologies and the community around them. The conference will be held in Calgary, Canada, August 24th - August 29th, 2020.

9.5 Governance and Community Participation

9.5.1 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.

9.5.2 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:

- Frank Warmerdam
- Daniel Morissette
- Howard Butler
- Tamas Szekeres
- Even Rouault (chair)
- Jukka Rahkonen
- Kurt Schwehr
- Norman Barker

- Sean Gillies
- Mateusz Łoskot

Past members:

- Andrey Kiselev (retired in 2019)

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.

HOW TO CONTRIBUTE?

10.1 Developer Contributions to GDAL

Install all required development packages: GNU make, g++, ...

Build:

```
cd gdal
./configure [options]
make -j8 -s
cd apps; make -s test_ogrsg; cd ..
```

Run command line utilities (without installing):

```
. scripts/setdevenv.sh
gdalinfo --version
```

Run autotest suite:

```
cd ../autotest
pip install -r requirements.txt
pytest
```

10.1.1 Git workflows with GDAL

This is not a git tutorial or reference manual by any means. This just collects a few best practice for git usage for GDAL development.

10.1.1.1 Commit message

Indicate a component name (eg a driver name), a short description and when relevant, a reference to a issue (with 'fixes #' if it actually fixes it)

```
COMPONENT_NAME: fix bla bla (fixes #1234)

Details here...
```

10.1.1.2 Initiate your work repository

Fork [OSGeo/GDAL](#) from GitHub UI, and then

```
git clone https://github.com/OSGeo/gdal
cd gdal
git remote add my_user_name https://github.com/my_user_name/gdal.git
```

10.1.1.3 Updating your local master against upstream master

```
git checkout master
git fetch origin
# Be careful: this will loose all local changes you might have done now
git reset --hard origin/master
```

10.1.1.4 Working with a feature branch

```
git checkout master


(potentially update your local master against upstream, as described above)


git checkout -b my_new_feature_branch

# do work. For example:
git add my_new_file
git add my_modifid_message
git rm old_file
git commit -a

# you may need to resynchronize against master if you need some bugfix
# or new capability that has been added since you created your branch
git fetch origin
git rebase origin/master

# At end of your work, make sure history is reasonable by folding non
# significant commits into a consistent set
git rebase -i master (use 'fixup' for example to merge several commits together,
and 'reword' to modify commit messages)

# or alternatively, in case there is a big number of commits and marking
# all them as 'fixup' is tedious
git fetch origin
git rebase origin/master
git reset --soft origin/master
git commit -a -m "Put here the synthetic commit message"

# push your branch
git push my_user_name my_new_feature_branch
From GitHub UI, issue a pull request
```

If the pull request discussion or Travis-CI/AppVeyor checks require changes, commit locally and push. To get a reasonable history, you may need to `git rebase -i master`, in which case you will have to force-push your branch with `git push -f my_user_name my_new_feature_branch`

10.1.1.5 Backporting bugfixes from master to a stable branch

```
git checkout master
With git log, identify the shalsum of the commit you want to backport
git checkout 2.2 (if you want to backport to 2.2)
git pull origin 2.2
(git checkout -b branch_name: if you intend to submit the backport as a pull request)
git cherry-pick the_shal_sum
git push ...
```

If changes are needed, do them and `git commit -a --amend`

10.1.1.6 Things you should NOT do

(For anyone with push rights to github.com/OSGeo/gdal) Never modify a commit or the history of anything that has been committed to <https://github.com/OSGeo/gdal>

10.2 Sphinx RST Style guide

This page contains syntax rules, tips, and tricks for using Sphinx and reStructuredText. For more information, please see this [comprehensive guide to reStructuredText](#), as well as the [Sphinx reStructuredText Primer](#).

10.2.1 Basic markup

10.2.2 Basic markup

A reStructuredText document is written in plain text. Without the need for complex formatting, one can be composed simply, just like one would any plain text document. For basic formatting, see this table:

Format	Syntax	Output
Italics	<code>*italics*</code> (single asterisk)	<i>italics</i>
Bold	<code>**bold**</code> (double asterisk)	bold
Monospace	<code>``monospace``</code> (double back quote)	monospace

Warning: Use of basic markup is **not recommend!** Where possible use sphinx inline directives to logically mark commands, parameters, options, input, and files. By using directives consistently these items can be styled appropriately.

10.2.3 Lists

There are two types of lists, bulleted lists and numbered lists. A **bulleted list** looks like this:

- An item
- Another item
- Yet another item

This is accomplished with the following code:

```
* An item
* Another item
* Yet another item
```

A **numbered list** looks like this:

1. First item
2. Second item
3. Third item

This is accomplished with the following code:

```
#. First item
#. Second item
#. Third item
```

Note that numbers are automatically generated, making it easy to add/remove items.

10.2.4 List-tables

Bulleted lists can sometimes be cumbersome and hard to follow. When dealing with a long list of items, use list-tables. For example, to talk about a list of options, create a table that looks like this:

Shapes	Description
Square	Four sides of equal length, 90 degree angles
Rectangle	Four sides, 90 degree angles

This is done with the following code:

```
.. list-table::
   :widths: 20 80
   :header-rows: 1

   * - Shapes
     - Description
   * - Square
     - Four sides of equal length, 90 degree angles
   * - Rectangle
     - Four sides, 90 degree angles
```

10.2.5 Page labels

Ensure every page has a label that matches the name of the file. For example if the page is named `foo_bar.rst` then the page should have the label:

```
.. _foo_bar:
```

Other pages can then link to that page by using the following code:

```
:ref:`foo_bar`
```

10.2.6 Linking

Links to other pages should never be titled as “here”. Sphinx makes this easy by automatically inserting the title of the linked document.

Bad More information about linking can be found *here*.

Good For more information, please see the section on *Linking*.

To insert a link to an external website:

```
`Text of the link <http://example.com>`__
```

The resulting link would look like this: [Text of the link](#)

Warning: It is very easy to have two links with the same text resulting in the following error:

```
** (WARNING/2) Duplicate explicit target name:foo**
```

To avoid these warnings use of a double __ generates an anonymous link.

10.2.7 Sections

Use sections to break up long pages and to help Sphinx generate tables of contents.

```
=====
Document title
=====
```

```
First level
-----
```

```
Second level
+++++++
```

```
Third level
*****
```

```
Fourth level
~~~~~
```

10.2.8 Notes and warnings

When it is beneficial to have a section of text stand out from the main text, Sphinx has two such boxes, the note and the warning. They function identically, and only differ in their coloring. You should use notes and warnings sparingly, however, as adding emphasis to everything makes the emphasis less effective.

Here is an example of a note:

Note: This is a note.

This note is generated with the following code:

```
.. note:: This is a note.
```

Similarly, here is an example of a warning:

```
Warning: Beware of dragons.
```

This warning is generated by the following code:

```
.. warning:: Beware of dragons.
```

10.2.9 Images

Add images to your documentation when possible. Images, such as screenshots, are a very helpful way of making documentation understandable. When making screenshots, try to crop out unnecessary content (browser window, desktop, etc). Avoid scaling the images, as the Sphinx theme automatically resizes large images. It is also helpful to include a caption underneath the image.:

```
.. figure:: image.png
   :align: center

   *Caption*
```

In this example, the image file exists in the same directory as the source page. If this is not the case, you can insert path information in the above command. The root / is the directory of the `conf.py` file.:

```
.. figure:: ../images/gdalicon.png
```

10.2.10 External files

Text snippets, large blocks of downloadable code, and even zip files or other binary sources can all be included as part of the documentation.

To include link to sample file, use the `download` directive:

```
:download:`An external file <example.txt>`
```

The result of this code will generate a standard link to an external file

To include a the contents of a file, use `literalinclude` directive:

Example of `:command:`gdalinfo`` use:

```
.. literalinclude:: example.txt
```

Example of `gdalinfo` use:

```
Driver: GTiff/GeoTIFF
Size is 512, 512
Coordinate System is:
PROJCS["NAD27 / UTM zone 11N",
    GEOGCS["NAD27",
        DATUM["North_American_Datum_1927",
```

(continues on next page)

(continued from previous page)

```

        SPHEROID["Clarke 1866",6378206.4,294.978698213901]],
        PRIMEM["Greenwich",0],
        UNIT["degree",0.0174532925199433]],
        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]]
Origin = (440720.000000,3751320.000000)
Pixel Size = (60.000000,-60.000000)
Corner Coordinates:
Upper Left  ( 440720.000, 3751320.000) (117d38'28.21"W, 33d54'8.47"N)
Lower Left  ( 440720.000, 3720600.000) (117d38'20.79"W, 33d37'31.04"N)
Upper Right ( 471440.000, 3751320.000) (117d18'32.07"W, 33d54'13.08"N)
Lower Right ( 471440.000, 3720600.000) (117d18'28.50"W, 33d37'35.61"N)
Center      ( 456080.000, 3735960.000) (117d28'27.39"W, 33d45'52.46"N)
Band 1 Block=512x16 Type=Byte, ColorInterp=Gray

```

The `literalinclude` directive has options for syntax highlighting, line numbers and extracting just a snippet:

```

Example of :command:`gdalinfo` use:

.. literalinclude:: example.txt
   :language: txt
   :linenos:
   :emphasize-lines: 2-6
   :start-after: Coordinate System is:
   :end-before: Origin =

```

10.2.11 Reference files and paths

Use the following syntax to reference files and paths:

```
:file:`myfile.txt`
```

This will output: `myfile.txt`.

You can reference paths in the same way:

```
:file:`path/to/myfile.txt`
```

This will output: `path/to/myfile.txt`.

For Windows paths, use double backslashes:

```
:file:`C:\\myfile.txt`
```

This will output: `C:\\myfile.txt`.

If you want to reference a non-specific path or file name:

```
:file:`{your/own/path/to}/myfile.txt`
```

This will output: `your/own/path/to/myfile.txt`

10.2.12 Reference commands

Reference commands (such as **gdalinfo**) with the following syntax:

```
:program:`gdalinfo`
```

Use option directive for command line options:

```
.. option:: -json
```

Display the output **in** json format.

Use describe to document create parameters:

```
.. describe:: WORLDFILE=YES
```

Force the generation of an associated ESRI world file (**with** the extension .wld).

11.1 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.”

11.2 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.

11.3 What does OGR stand for?

OGR used to stand for OpenGIS Simple Features Reference Implementation. However, since OGR is not fully compliant with the OpenGIS Simple Feature specification and is not approved as a reference implementation of the spec the name was changed to OGR Simple Features Library. The only meaning of OGR in this name is historical. OGR is also the prefix used everywhere in the source of the library for class names, filenames, etc.

11.4 What does CPL stand for?

Common Portability Library. Think of it as GDAL internal cross-platform standard library. Back in the early days of GDAL development, when cross-platform development as well as compatibility and standard conformance of compilers was a challenge (or PITA), CPL proved necessary for smooth portability of GDAL/OGR.

CPL, or parts of it, is used by some projects external to GDAL (eg. MITAB, libgeotiff).

11.5 When was the GDAL project started?

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

11.6 Is GDAL/OGR proprietary software?

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

11.7 What license does GDAL/OGR use?

See *License*

11.8 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.

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

See *Software using GDAL*

11.9.1 Software using GDAL

- **3D DEM Viewer** from MS MacroSystem
- **Biodiverse** A tool for the spatial analysis of diversity. Uses GDAL for import/export of data.
- **Bluemapia** Multi-Map(Google,Microsoft,Open Street Map, NOAA/BSB Charts,self-calibrated raster) location-based GPS app for Windows Mobile.
- **BRL-CAD** An open source solid modeling computer-aided design system.
- **Cadcorp SIS**: A Windows GIS with a GDAL and OGR plugins.
- **CARTO** A cloud mapping platform to analyze and visualize geospatial data.
- **Cartographica** Macintosh GIS package.
- **CatchmentSIM** A Windows terrain analysis model for hydrologic applications.
- **Daylon Leveller** A terrain/heightfield/bumpmap modeler
- **Demeter** Another OpenGL based terrain engine somewhat similar to VTP.
- **Enfusion** Analysis and visualization of time-varying spatial datasets integrated via true data fusion.
- **EOxServer** OGC-compliant server for Earth Observation (EO) data supporting WMS and WCS with EO application profiles.
- **ERDAS ER Viewer** Image viewer for very large JPEG 2000 and ECW files.
- **ESRI ArcGIS 9.2+** A popular GIS platform.

- [Eternix Blaze](#) Advanced geo-spatial visualization application and SDK.
- [exactextract](#) Fast and accurate raster/vector zonal statistics.
- [FalconView](#) Windows-based GIS platform with roots in military mission planning, now available as a free GIS visualization and analysis package.
- [Feature Data Objects \(FDO\)](#) Open source spatial data access libraries.
- [Fiona](#) Fiona is OGR's neater API – sleek and elegant on the outside, indomitable power on the inside.
- [flighttrack](#) GPS track viewing and downloading software for Mac.
- [FME](#) A GIS translator package includes a GDAL plugin.
- [GdalToTiles](#) C# Program (open source) for making image tiles for Google Earth with KML Superoverlay.
- [GenGIS](#) Software for geospatial analysis of genetic data.
- [Geographic Imager](#) DEM / aerial / satellite image processing GIS plug-in for Adobe Photoshop, by Avenza Systems.
- [GeoDa](#) Introduction to Spatial Data Analysis (spatial autocorrelation and spatial regression)
- [GeoDjango](#) A framework for building geographic web applications.
- [GeoDMS](#) A framework for building spatial calculation models.
- [GeoKettle](#) An open source spatial ETL (Extract, Transform and Load) tool.
- [GeoFusion](#) 3D visualization.
- [GeoNotebook](#) a Jupyter notebook extension for geospatial visualization and analysis.
- [GeoServer](#) a open source software server written in Java that allows users to share and edit geospatial data.
- [GeoView Pro](#) IOS mobile mapping application.
- [Geoweb3d](#) A 3D virtual globe that provides on-the-fly, game-quality visualization of GIS data.
- [GMT \(Generic Mapping Tools\)](#) an open source collection of tools for processing and displaying xy and xyz datasets.
- [Google Earth](#) A 3D world viewer.
- [GPSeismic](#) A suite of applications for seismic survey.
- [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.
- [gvSIG](#) Desktop GIS Client.
- [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.
- [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.
- [Infraworks](#) a BIM software for infrastructure project design, part of the Autodesk suite.
- [iShare](#) Web data integration and publishing platform by Astun Technology.

- [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.
- [Makai Voyager](#) An advanced 3D/4D geospatial visualization platform.
- [MapGuide](#) Open source web mapping server.
- [MapInfo Professional](#) Desktop GIS and mapping application
- [Mapnik](#) C++/ Python mapping toolkit
- [MapServer](#) A popular web mapping application with GDAL support.
- [MapTiler](#) Generator of tiles for interactive maps and overlays made from raster images and geodata.
- [Maptitude Mapping Software](#) Desktop GIS and business mapping application
- [MapWindow](#) open source ActiveX control with GIS functionality.
- [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 Converter](#) Online converter of GDAL raster and OGR vector formats
- [NASA Ames Stereo Pipeline](#) Software for creating terrain models and ortho images from planetary stereo images.
- [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
- [OPALS](#) Orientation and Processing of Airborne Laser Scanning Data
- [OpenCPN](#) A concise ChartPlotter/Navigator. A cross-platform ship-borne GUI application.
- [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.

- [Procura](#) Landholding inspection system developed for the UK Homes and Communities Agency. GDAL is used for checking out background mapping.
- [PYXIS](#) An application for viewing performing analysis and modeling on user's geo-located data.
- [QGIS](#) A cross platform desktop GIS.
- [QLandkarte GT](#) GT is the ultimate outdoor aficionado's tool for GPS maps in GeoTiff format as well as Garmin's img vector map format.
- [R](#) A free software environment for statistical computing and graphics, with bindings to GDAL via the rgdal package.
- [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.
- [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.
- [SNAP](#) Sentinel Application Platform for Earth Observation processing and analysis.
- [SpacEyes3D](#) 3D visualization software for cartographic data.
- [Spatial Manger](#) A product suite designed designed to manage spatial data in a simple, fast and inexpensive way. Uses GDAL to import/export data.
- [Carmenta Engine](#) (previously known as SpatialAce): A GIS Rapid Application Development environment
- [StarSpan](#) raster/vector analysis.
- [TacitView](#) An imagery visualization and exploitation package for military intelligence.
- [TatukGIS](#) Desktop GIS mapping and data editing application.
- [TerraGo Technologies](#) The GeoPDF file format is used to distribute and collaborate geospatial data and uses GDAL for data import/export.
- [t-rex](#) Vector tile server written in Rust.
- [TerrainView](#) Interactive real-time 3D GIS Software.
- [Thuban](#) A multi-platform interactive geographic data viewer.
- [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.
- [Virtual Terrain Project](#) fostering tools for easy construction of the real world in interactive, 3D digital form.
- [WindNinja](#) wind model for fire behavior modeling.

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

GDAL/OGR can be compiled with a C++11 capable compiler.

11.11 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.

11.12 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*

11.13 How do I cite GDAL ?

See *CITATION*

12.1 License

In general GDAL/OGR is licensed under an MIT/X style license with the following terms:

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.

The full licensing terms are available in the [LICENSE.TXT](#) file.

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

Symbols

- 3d
 - command line option, 25
 - gdal_translate command line option, 38
- 8
 - gdaladdo command line option, 52
- A <filename>
 - command line option, 65
- V
 - gdalsrsinfo command line option, 57
- A_band=<n>
 - command line option, 65
- NoDataValue=<value>
 - command line option, 65
- allBands=[A-Z]
 - command line option, 65
- bingkey=<BINGKEY>
 - gdal_translate command line option, 37
- calc=expression
 - command line option, 65
- cflags
 - command line option, 68
- co=<option>
 - command line option, 65
- config <key> <value>
 - command line option, 5, 80
- copyright=<COPYRIGHT>
 - gdal_translate command line option, 37
- creation-option=<option>
 - command line option, 65
- debug
 - command line option, 66
- debug <value>
 - command line option, 5, 80
- force-kml
 - gdal_translate command line option, 36
- format <format>
 - command line option, 5, 80
- format=<gdal_format>
 - command line option, 65
- formats
 - command line option, 5, 68, 80
 - ogrinfo command line option, 82
- googlekey=<GOOGLEKEY>
 - gdal_translate command line option, 37
- help
 - command line option, 65
 - gdal_translate command line option, 36
- help-general
 - command line option, 5, 63, 80, 92
- libs
 - command line option, 68
- no-kml
 - gdal_translate command line option, 36
- ogr-enabled
 - command line option, 68
- optfile <filename>
 - command line option, 5, 80
- outfile=<filename>
 - command line option, 65
- overwrite
 - command line option, 66
- prefix
 - command line option, 68
- processes=<NB_PROCESSES>
 - gdal_translate command line option, 36
- profile=<PROFILE>
 - gdal_translate command line option, 35
- quiet
 - command line option, 66
 - gdal_translate command line option, 36
- resampling=<RESAMPLING>
 - gdal_translate command line option, 36

```

--resume
    gdal_translate command line option,
    36
--s_srs=<SRS>
    gdal_translate command line option,
    36
--single-line
    gdalsrsinfo command line option, 57
--srcnodata=<NODATA>
    gdal_translate command line option,
    36
--tilesize=<TILESIZE>
    gdal_translate command line option,
    36
--title=<TITLE>
    gdal_translate command line option,
    37
--type=<datatype>
    command line option, 65
--url=<URL>
    gdal_translate command line option,
    36
--verbose
    gdal_translate command line option,
    36
--version
    command line option, 5, 68, 80
    gdal_translate command line option,
    36
--webviewer=<WEBVIEWER>
    gdal_translate command line option,
    37
--xyz
    gdal_translate command line option,
    36
--zoom=<ZOOM>
    gdal_translate command line option,
    36
-a <NODATA>
    gdal_translate command line option,
    36
-a <attribute_name>
    gdal_translate command line option,
    38
-a <name>
    command line option, 25
-a <[algorithm[:parameter1=value1][:parameter2=value2]...]>
    gdaladdo command line option, 46
-a_nodata <output_nodata_value>
    gdaladdo command line option, 34
-a_nodata <value>
    command line option, 64
    gdal_translate command line option,
    12, 38
    gdal_viewshed command line option,
    71
-a_offset<value>
    gdal_translate command line option,
    11
-a_scale <value>
    gdal_translate command line option,
    11
-a_srs <srs_def>
    command line option, 63
    gdal_translate command line option,
    11, 38
    gdaladdo command line option, 46
    gdalbuildvrt command line option, 24
    ogr2ogr command line option, 86
    ogrmerge.py command line option, 95
-a_srs <srs>
    command line option, 6
-a_ullr <ulx> <uly> <lrx> <lry>
    gdal_translate command line option,
    11
-a_ullr ulx uly lrx lry:
    command line option, 63
-a_ulurll ulx uly urx ury llx lly:
    command line option, 63
-accept_different_schemas
    ogrtindex command line option, 91
-add
    gdal_translate command line option,
    38
-addalpha
    gdalbuildvrt command line option, 23
-addfields
    ogr2ogr command line option, 89
-al
    ogrinfo command line option, 81
-allow_projection_difference
    gdalbuildvrt command line option, 24
-alo NAME=VALUE
    gnmanalyse command line option, 99
-alpha
    command line option, 30
-alt <altitude>
    command line option, 29
-amax <name>
    command line option, 25
-amax <name>
    command line option, 25
-append
    ogr2ogr command line option, 85
    ogrmerge.py command line option, 95
-approx_stats
    command line option, 63
    gdalinfo command line option, 7

```

```

-array {array_name}
    gdalinfo command line option, 75
-array <array_spec>
    gdalmdimtranslate command line
        option, 78
-arrayoption <NAME=VALUE>
    gdalinfo command line option, 75
-at
    gdal_translate command line option,
        38
-az <azimuth>
    command line option, 29
-b <BINGKEY>
    gdal_translate command line option,
        37
-b <band>
    command line option, 25, 28, 66
    gdal_translate command line option,
        10, 37
    gdal_viewshed command line option,
        71
    gdaladdo command line option, 13, 33,
        53
    gdalbuildvrt command line option, 23
    gdallocationinfo command line
        option, 55
-b band
    command line option, 54
-bitdepth <val>
    command line option, 67
-bl <gfid>
    gnmmanage command line option, 98
-burn <value>
    gdal_translate command line option,
        38
-c <COPYRIGHT>
    gdal_translate command line option,
        37
-c <cost>
    gnmmanage command line option, 97
-cblend <distance>
    gdalwarp command line option, 19
-cc <value>
    gdal_viewshed command line option,
        72
-checksum
    gdalinfo command line option, 7
-cl <layername>
    gdalwarp command line option, 19
-clean
    gdaladdo command line option, 14
-clipdst <xmin> <ymin> <xmax> <ymax>
    ogr2ogr command line option, 87
-clipdstlayer <layername>
    ogr2ogr command line option, 87
-clipdstsql <sql_statement>
    ogr2ogr command line option, 87
-clipdstwhere <expression>
    ogr2ogr command line option, 87
-clipsrc [xmin ymin xmax
        ymax] | WKT | datasource | spat_extent
    gdaladdo command line option, 46
    ogr2ogr command line option, 87
-clipsrclayer <layername>
    gdaladdo command line option, 47
    ogr2ogr command line option, 87
-clipsrcsql <sql_statement>
    gdaladdo command line option, 46
    ogr2ogr command line option, 87
-clipsrcwhere <expression>
    gdaladdo command line option, 47
    ogr2ogr command line option, 87
-co "NAME=VALUE"
    ogrtindex command line option, 42
-co <NAME=VALUE>
    command line option, 6, 28, 67
    gdal_translate command line option,
        12, 39, 44
    gdal_viewshed command line option,
        71
    gdaladdo command line option, 34, 47,
        51
    gdalmdimtranslate command line
        option, 78
    gdalwarp command line option, 19
-color <c1,c2,c3...cn>
    ogrtindex command line option, 43
-colorinterp <red|green|blue|alpha|gray|undefined|...>
    gdal_translate command line option,
        12
-colorinterp_X <red|green|blue|alpha|gray|undefined|...>
    gdal_translate command line option,
        12
-combined
    command line option, 29
-compute_edges
    command line option, 28
-create
    command line option, 93
-createonly
    gdaladdo command line option, 35
-crop_to_cutline
    gdalwarp command line option, 19
-csql <query>
    gdalwarp command line option, 19
-csv <csvFileName>
    gdal_translate command line option,
        45

```

-csvDelim <column delimiter>
gdal_translate command line option, 45

-ct <string>
gdaltransform command line option, 40
gdalwarp command line option, 16
ogr2ogr command line option, 86

-cutline <datasource>
gdalwarp command line option, 19

-cvmid <meta_conflict_value>
gdalwarp command line option, 19

-cwhere <expression>
gdalwarp command line option, 19

-d <ds_name>
gnmanalyse command line option, 99

-datelineoffset
ogr2ogr command line option, 87

-detailed
gdalinfo command line option, 75

-dialect <dialect>
gdal_translate command line option, 38
ogr2ogr command line option, 85
ogrinfo command line option, 81

-dim <val>
ogr2ogr command line option, 86

-dir <dir>
gnmmanage command line option, 97

-distunits PIXEL|GEO
gdaladdo command line option, 52

-doo NAME=VALUE
ogr2ogr command line option, 87

-doo <NAME=VALUE>
gdalwarp command line option, 19

-ds_transaction
ogr2ogr command line option, 87

-dsco NAME=VALUE
gnmanalyse command line option, 99
gnmmanage command line option, 97
ogr2ogr command line option, 86

-dsco <NAME=VALUE>
command line option, 25, 93
ogrmerge.py command line option, 95

-dstalpha
gdalwarp command line option, 18

-dstband <n>
gdaladdo command line option, 51

-dstnodata <value [value...]>
gdalwarp command line option, 18

-e
gdal_translate command line option, 36
gdalsrsinfo command line option, 57

-e <base>
command line option, 26

-eco
gdal_translate command line option, 11

-epo
gdal_translate command line option, 11

-et <err_threshold>
gdalwarp command line option, 17

-et <max_pixel_err>
command line option, 62

-exact_color_entry
command line option, 30

-expand gray|rgb|rgba
gdal_translate command line option, 10

-explodecollections
ogr2ogr command line option, 88

-exponent <exp_val>
gdal_translate command line option, 11

-f <ds_format>
gnmanalyse command line option, 99

-f <format_name>
command line option, 92
gnmmanage command line option, 97
ogr2ogr command line option, 85

-f <format>
command line option, 25, 69
ogrmerge.py command line option, 95
ogrindex command line option, 21

-f <ogr_format>
gdaladdo command line option, 53

-f <output_format>
ogrindex command line option, 91

-fid <fid>
ogrinfo command line option, 81

-fid fid
ogr2ogr command line option, 87

-fieldTypeToString type1,...
ogr2ogr command line option, 88

-field_strategy FirstLayer|Union|Intersection
ogrmerge.py command line option, 96

-fieldmap
ogr2ogr command line option, 88

-fields YES|NO:
ogrinfo command line option, 81

-fixed-buf-val <n>
gdaladdo command line option, 52

-fl <level>
command line option, 26

-forceNullable
ogr2ogr command line option, 89

```

-g <GOOGLEKEY>
    gdal_translate command line option,
    37
-gcp <pixel> <line> <easting>
    <northing> <elevation>
    gdal_translate command line option,
    12
-gcp <pixel> <line> <easting>
    <northing> [<elevation>]
    gdaltransform command line option,
    41
-gcp <ungeoref_x> <ungeoref_y>
    <georef_x> <georef_y>
    <elevation>
    ogr2ogr command line option, 88
-gcp pixel line easting northing
    [elevation]
    command line option, 64
-geoloc
    gdallocationinfo command line
    option, 55
    gdaltransform command line option,
    41
    gdalwarp command line option, 17
-geom YES|NO|SUMMARY|WKT|ISO_WKT
    ogrinfo command line option, 81
-geomfield <field>
    ogr2ogr command line option, 86
    ogrinfo command line option, 81
-get_coord
    command line option, 93
-get_pos
    command line option, 93
-get_subline
    command line option, 94
-group <group_spec>
    gdalmdimtranslate command line
    option, 78
-gt n
    ogr2ogr command line option, 87
-h
    command line option, 65
    gdal_translate command line option,
    36
-hiddenodata
    gdalbuildvrt command line option, 23
-hist
    gdalinfo command line option, 7
-i
    gdal_translate command line option,
    37
    gdaltransform command line option,
    41
-i <interval>
    command line option, 26
-ic <invcost>
    gnmmanage command line option, 97
-igor
    command line option, 29
-info
    gnmmanage command line option, 97
-init <"value(s)">
    gdaladdo command line option, 35
-init <value>
    gdal_translate command line option,
    38
-inodata
    command line option, 25
-input_file_list <mylist.txt>
    gdalbuildvrt command line option, 24
-iv <value>
    gdal_viewshed command line option,
    72
-json
    gdalinfo command line option, 7
-k
    gdal_translate command line option,
    36
-l <layer_name>
    gnanalyse command line option, 99
-l <layername>
    gdal_translate command line option,
    38
    gdaladdo command line option, 47
-l <src_line_datasource_name>
    command line option, 93
-l layer_name
    gnmmanage command line option, 97
-l_srs <srs_def>
    gdallocationinfo command line
    option, 55
-lco NAME=VALUE
    gnanalyse command line option, 99
    ogr2ogr command line option, 86
-lco <NAME=VALUE>
    command line option, 26, 93
    ogrmerge.py command line option, 95
-levels <numberOfLevels>
    gdal_translate command line option,
    44
-lf <field_name>
    command line option, 93
-lifonly
    gdallocationinfo command line
    option, 55
-limit {number}
    gdalinfo command line option, 75
-limit nb_features

```

```

    ogr2ogr command line option, 87
-listmdd
    gdalinfo command line option, 8
    ogrinfo command line option, 81
-ln <layer_name>
    command line option, 93
-lname <name>
    ogrtindex command line option, 91
-lnum <n>
    ogrtindex command line option, 91
-lyr_name <name>
    ogrtindex command line option, 21
-m <position>
    command line option, 93
-makevalid
    ogr2ogr command line option, 88
-mapFieldType srctype|All=dsttype,...
    ogr2ogr command line option, 88
-mask <band>
    gdal_translate command line option,
    10
-mask <filename>
    gdaladdo command line option, 52
-mask filename
    command line option, 54
-maxdist <n>
    gdaladdo command line option, 52
-maxsubfields <val>
    ogr2ogr command line option, 88
-mb <position>
    command line option, 94
-md <value>
    gdal_viewshed command line option,
    72
-md max_distance
    command line option, 54
-mdd <domain>
    ogrinfo command line option, 82
-mdd <domain>|all
    gdalinfo command line option, 8
-me <position>
    command line option, 94
-minsize <val>
    gdaladdo command line option, 14
-mm
    gdalinfo command line option, 7
-mo META-TAG=VALUE
    command line option, 64
    gdal_translate command line option,
    12
    ogr2ogr command line option, 89
-multi
    gdalwarp command line option, 19
-multidirectional
    command line option, 29
-n
    gdal_translate command line option,
    36
-n <color>
    gdaladdo command line option, 32
-n <nodata_value>
    gdaladdo command line option, 34
-nb <non_black_pixels>
    ogrtindex command line option, 43
-near <dist>
    ogrtindex command line option, 43
-nearest_color_entry
    command line option, 30
-nln <layer_name_template>
    ogrmerge.py command line option, 95
-nln <name>
    command line option, 26
    ogr2ogr command line option, 86
-nlt <type>
    ogr2ogr command line option, 86
-noNativeData
    ogr2ogr command line option, 89
-nocount
    ogrinfo command line option, 82
-noct
    gdalinfo command line option, 7
-nodata <n>
    gdaladdo command line option, 52
-nodata <val>
    command line option, 67
-noextent
    ogrinfo command line option, 82
-nofl
    gdalinfo command line option, 8
-nogcp
    gdal_translate command line option,
    12
    gdalinfo command line option, 7
-nogeomtype
    ogrinfo command line option, 82
-nomask
    command line option, 54
    gdaladdo command line option, 52
-nomd
    gdalinfo command line option, 7
    gdalwarp command line option, 19
    ogr2ogr command line option, 89
    ogrinfo command line option, 81
-nopretty
    gdalinfo command line option, 75
-norat
    gdal_translate command line option,
    12

```

```

    gdalinfo command line option, 7
-nosrcalpha
    gdalwarp command line option, 18
-novshiftgrid
    gdalwarp command line option, 16
-o <dst_datasource_name>
    command line option, 93
-o <out_dsname>
    ogrmerge.py command line option, 94
-o <out_filename>
    gdaladdo command line option, 34
-o <out_type>
    gdalsrsinfo command line option, 57
-o <outfile>
    ogrtindex command line option, 42
-o name=value
    command line option, 54
-of <field_name>
    command line option, 93
-of <format>
    command line option, 6, 28
    gdal_translate command line option,
        10, 38, 44
    gdaladdo command line option, 32-34,
        46, 51
    gdalmdimtranslate command line
        option, 78
    gdalwarp command line option, 19
    ogrtindex command line option, 42
-of <format>:
    command line option, 66
-of format
    command line option, 54
-off <offset>
    command line option, 26
-offset <value>
    command line option, 64
-om <output mode>
    gdal_viewshed command line option,
        72
-on <layer_name>
    command line option, 93
-oo NAME=VALUE
    command line option, 64
    gdal_translate command line option,
        12
    gdaladdo command line option, 14
    gdalbuildvrt command line option, 24
    gdallocationinfo command line
        option, 55
    ogr2ogr command line option, 87
    ogrinfo command line option, 81
-oo <NAME=VALUE>
    gdalinfo command line option, 8, 75
    gdalwarp command line option, 19
-optim {[AUTO]/VECTOR/RASTER}}
    gdal_translate command line option,
        39
    gdalbuildvrt command line option, 24
-order <n>
    gdaltransform command line option,
        41
    gdalwarp command line option, 17
    ogr2ogr command line option, 88
-ot <type>
    gdal_translate command line option,
        10, 39, 44
    gdaladdo command line option, 34, 46,
        52
    gdalwarp command line option, 18
-output_xy
    gdaltransform command line option,
        41
-outsize <xsize ysize>
    gdaladdo command line option, 46
-outsize <xsize>[%]|0 <ysize>[%]|0
    gdal_translate command line option,
        10
-ov <value>
    gdal_viewshed command line option,
        72
-overlap< <val_in_pixel>
    gdal_translate command line option,
        44
-overview <overview_level>
    gdallocationinfo command line
        option, 55
-overwrite
    gdalbuildvrt command line option, 24
    gdalwarp command line option, 19
    ogr2ogr command line option, 85
-overwrite_ds
    ogrmerge.py command line option, 95
-overwrite_layer
    ogrmerge.py command line option, 95
-ovr <level|AUTO|AUTO-n|NONE>
    gdalwarp command line option, 17
-ox <value>
    gdal_viewshed command line option,
        71
-oy <value>
    gdal_viewshed command line option,
        72
-oz <value>
    gdal_viewshed command line option,
        72
-p
    command line option, 26, 29

```

-p <PROFILE>
 gdal_translate command line option, 35
 -p <src_repers_datasource_name>
 command line option, 93
 -pct
 gdaladdo command line option, 34
 -pct <palette_file>
 gdaladdo command line option, 32
 -pf <field_name>
 command line option, 93
 -pm <pos_field_name>
 command line option, 93
 -pn <layer_name>
 command line option, 93
 -preserve_fid
 ogr2ogr command line option, 87
 -progress
 command line option, 92
 ogr2ogr command line option, 85
 ogrmerge.py command line option, 96
 -proj4
 gdalinfo command line option, 8
 -projwin <ulx> <uly> <lrx> <lry>
 gdal_translate command line option, 11
 -projwin_srs <srs_def>
 gdal_translate command line option, 11
 -ps <pixelsize_x> <pixelsize_y>
 gdal_translate command line option, 44
 gdaladdo command line option, 34
 -pyramidOnly
 gdal_translate command line option, 44
 -q
 command line option, 26, 28, 54, 67
 gdal_translate command line option, 12, 36, 39
 gdaladdo command line option, 47, 53
 gdalbuildvrt command line option, 24
 gdalwarp command line option, 19
 ogrinfo command line option, 81
 ogrindex command line option, 43
 -quiet
 command line option, 92
 -r
 command line option, 69
 -r {nearest (default), average, gauss, cubic, cubicspline, lanczos, average, mode}
 gdaladdo command line option, 13
 -r {nearest (default), bilinear, cubic, cubicspline, lanczos, average, mode}
 gdal_translate command line option, 10
 gdalbuildvrt command line option, 24
 -r {nearest, bilinear, cubic
 (default), cubicspline, lanczos, average}
 command line option, 67
 -r <RESAMPLING>
 gdal_translate command line option, 36
 -r <algorithm>
 gdal_translate command line option, 45
 -r <resampling_method>
 gdalwarp command line option, 18
 -r <src_parts_datasource_name>
 command line option, 93
 -refine_gcps <tolerance minimum_gcps>
 gdalwarp command line option, 17
 -relaxedFieldNameMatch
 ogr2ogr command line option, 89
 -resolution {highest|lowest|average|user}
 gdalbuildvrt command line option, 23
 -resume
 gdal_translate command line option, 45
 -rgba
 gdaladdo command line option, 33
 -rl
 ogrinfo command line option, 81
 -rn <layer_name>
 command line option, 93
 -ro
 command line option, 63
 gdaladdo command line option, 13
 ogrinfo command line option, 81
 -rpc
 gdaltransform command line option, 41
 gdalwarp command line option, 17
 -s <SRS>
 gdal_translate command line option, 36
 -s <scale>
 command line option, 29
 -s <step>
 command line option, 93
 -s_srs <srs def>
 gdalwarp command line option, 16
 -s_srs <srs_def>
 gdal_translate command line option, 45
 -s_srs {nearest, bilinear, cubic, cubicspline, lanczos, average, mode}
 ogrmerge.py command line option, 95
 -s_srs <srs_defn>

```

    command line option, 62
-s_srs <srs>
    command line option, 6
-scale <value>
    command line option, 64
-scale [src_min src_max [dst_min
    dst_max]]
    gdal_translate command line option,
    10
-scaleaxes <scaleaxes_spec>
    gdalmdimtranslate command line
    option, 79
-sd <n>
    gdalinfo command line option, 8
-sd< <subdataset>
    gdalbuildvrt command line option, 23
-sds
    gdal_translate command line option,
    12
    gdalinfo command line option, 70
-segmentize <max_dist>
    ogr2ogr command line option, 88
-select <field_list>
    ogr2ogr command line option, 85
-separate
    gdaladdo command line option, 34
    gdalbuildvrt command line option, 24
-setalpha
    ogrtindex command line option, 43
-setci
    gdalwarp command line option, 19
-setmask
    ogrtindex command line option, 43
-setstatsmin max mean stddev
    command line option, 63
-si smooth_iterations
    command line option, 54
-simplify <tolerance>
    ogr2ogr command line option, 88
-single
    ogrmerge.py command line option, 95
-skip_different_projection
    ogrtindex command line option, 21, 91
-skipfailures
    ogr2ogr command line option, 85
    ogrmerge.py command line option, 96
-snodata <value>
    command line option, 25
-so
    ogrinfo command line option, 81
-spat <xmin> <ymin> <xmax> <ymax>
    gdaladdo command line option, 46
    ogr2ogr command line option, 85
    ogrinfo command line option, 81
    -spat_adjust {union(default), intersection, none, none}
        command line option, 67
-spat_srs <srs_def>
    ogr2ogr command line option, 86
-splitlistfields
    ogr2ogr command line option, 88
-sql <select_statement>
    gdal_translate command line option,
    38
    gdaladdo command line option, 47
-sql <sql_statement>
    ogr2ogr command line option, 85
-sql <statement>
    ogrinfo command line option, 81
-src_geom_type <geom_type_name[,geom_type_name]*>
    ogrmerge.py command line option, 95
-src_layer_field_content
    <layer_name_template>
    ogrmerge.py command line option, 96
-src_layer_field_name <name>
    ogrmerge.py command line option, 96
-src_srs_format <format>
    ogrtindex command line option, 91
-src_srs_format <type>
    ogrtindex command line option, 21
-src_srs_name <field_name>
    ogrtindex command line option, 21, 91
-srcalpha
    gdalwarp command line option, 18
-srcband <n>
    gdaladdo command line option, 51
-srcnodata <value [value...]>
    gdalwarp command line option, 18
-srcnodata <value> [<value>...]
    gdalbuildvrt command line option, 23
-srcwin <xoff> <yoff> <xsize> <ysize>
    gdal_translate command line option,
    11
-stats
    command line option, 63
    gdal_translate command line option,
    12
    gdalinfo command line option, 7
-strict
    gdal_translate command line option,
    10
-subset <subset_spec>
    gdalmdimtranslate command line
    option, 79
-t <TITLE>
    gdal_translate command line option,
    37
-t_srs <srs_def>

```

gdaltransform command line option, 40
 gdalwarp command line option, 16
 ogr2ogr command line option, 86
 ogrmerge.py command line option, 95
 -t_srs <srs_defn>
 command line option, 62
 -t_srs <srs_name>
 gnmmanage command line option, 97
 -t_srs <srs>
 command line option, 6
 -t_srs <target_srs>
 ogrindex command line option, 91
 -t_srs <target_srs>:
 ogrindex command line option, 21
 -tap
 gdal_translate command line option, 39
 gdaladdo command line option, 34
 gdalbuildvrt command line option, 23
 gdalwarp command line option, 17
 -targetDir <directory>
 gdal_translate command line option, 44
 -te <xmin ymin xmax ymax>
 gdalwarp command line option, 17
 -te <xmin> <ymin> <xmax> <ymax>
 gdal_translate command line option, 39
 -te xmin ymin xmax ymax
 gdalbuildvrt command line option, 23
 -te_srs <srs_def>
 gdalwarp command line option, 17
 -threads {ALL_CPUS,number}
 command line option, 67
 -tileIndex <tileIndexName>
 gdal_translate command line option, 45
 -tileIndexField <tileIndexFieldName>
 gdal_translate command line option, 45
 -tileindex
 gdalbuildvrt command line option, 23
 -tileindex <field_name>
 ogrindex command line option, 21, 91
 -to NAME=VALUE
 gdal_translate command line option, 38
 gdaltransform command line option, 41
 -to <NAME=VALUE>
 gdalwarp command line option, 16
 -tps
 gdaltransform command line option, 41
 gdalwarp command line option, 17
 ogr2ogr command line option, 88
 -tr <res> <yres>
 gdalbuildvrt command line option, 23
 -tr <xres> <yres>
 command line option, 63
 gdal_translate command line option, 10, 39
 gdalwarp command line option, 17
 -trigonometric
 command line option, 30
 -ts <width> <height>
 gdal_translate command line option, 39
 gdalwarp command line option, 17
 -txe <xmin> <xmax>
 gdaladdo command line option, 46
 -tye <ymin> <ymax>
 gdaladdo command line option, 46
 -tz <value>
 gdal_viewshed command line option, 72
 -u
 command line option, 69
 -u <URL>
 gdal_translate command line option, 36
 -ul_lr <ulx> <uly> <lrx> <lry>
 gdaladdo command line option, 34
 -unbl <gfid>
 gnmmanage command line option, 98
 -unblall
 gnmmanage command line option, 98
 -units <value>
 command line option, 64
 -unscale
 gdal_translate command line option, 11
 -unsetDefault
 ogr2ogr command line option, 89
 -unsetFid
 ogr2ogr command line option, 89
 -unsetFieldWidth
 ogr2ogr command line option, 88
 -unsetgt
 command line option, 63
 -unsetmd
 command line option, 64
 -unsetnodata
 command line option, 64
 -unsetrpc
 command line option, 63

-unsetstats
 command line option, 63

-update
 ogr2ogr command line option, 85
 ogrmerge.py command line option, 95

-useDirForEachRow
 gdal_translate command line option, 45

-use_input_nodata YES/NO
 gdaladdo command line option, 52

-v
 gdal_translate command line option, 36, 44
 gdaladdo command line option, 34

-valonly
 gdallocationinfo command line option, 55

-values <n>, <n>, <n>
 gdaladdo command line option, 52

-vrt_nodata <value> [<value>...]
 gdalbuildvrt command line option, 24

-vv <value>
 gdal_viewshed command line option, 72

-w <WEBVIEWER>
 gdal_translate command line option, 37

-w <weight_val>
 command line option, 66

-wgs84
 gdallocationinfo command line option, 55

-where <expression>
 gdal_translate command line option, 38
 gdaladdo command line option, 47

-where <restricted_where>
 ogrinfo command line option, 81

-where restricted_where
 ogr2ogr command line option, 85

-white
 ogrindex command line option, 43

-wkt_format WKT1|WKT2|WKT2_2015|WKT2_2018
 gdalinfo command line option, 8

-wkt_format <format>
 ogrinfo command line option, 82

-wm <memory_in_mb>
 gdalwarp command line option, 18

-wo "NAME=VALUE"
 gdalwarp command line option, 17

-wrapdateline
 ogr2ogr command line option, 87

-write_absolute_path
 ogrindex command line option, 21, 91

-wt <type>
 gdalwarp command line option, 18

-x <long>
 command line option, 93

-xml
 gdallocationinfo command line option, 55

-y <lat>
 command line option, 93

-z <ZOOM>
 gdal_translate command line option, 36

-z <factor>
 command line option, 29

-z_increase <increase_value>
 gdaladdo command line option, 46

-z_multiply <multiply_value>
 gdaladdo command line option, 46

-zero_for_flat
 command line option, 30

-zfield <field_name>
 gdaladdo command line option, 46
 ogr2ogr command line option, 88

<datasetname>
 command line option, 69

<datasource_name>
 ogrinfo command line option, 82

<dest_file>
 gdaladdo command line option, 32, 33

<dst_dataset>
 gdal_translate command line option, 12
 gdalmdimtranslate command line option, 79

<dst_filename>
 gdal_translate command line option, 39
 gdaladdo command line option, 47

<dstfile>
 gdaladdo command line option, 51
 gdaltransform command line option, 41

<fieldname>
 gdalwarp command line option, 20

<filename>
 gdaladdo command line option, 53

<gdal_file>
 gdaladdo command line option, 14

<gdal_file>
 ogrindex command line option, 21

<gnm_name>
 gnmanalyse command line option, 99
 gnmmanage command line option, 98

<golden_file>
 gdalinfo command line option, 70

<infile>
 ogrindex command line option, 43
 <layer>
 gdaladdo command line option, 53
 gnmmmanage command line option, 98
 ogrinfo command line option, 82
 <levels>
 gdaladdo command line option, 14
 <mode>
 command line option, 27, 69
 <new_file>
 gdalinfo command line option, 70
 <newdatasetname>
 command line option, 69
 <out_dataset>
 command line option, 67
 <out_file>
 gdaladdo command line option, 53
 <pan_dataset>
 command line option, 67
 <raster_file>
 gdaladdo command line option, 53
 <source_file>
 gdaladdo command line option, 32, 33
 <spectral_dataset>[,band=num]
 command line option, 67
 <src_dataset>
 gdal_translate command line option, 12
 gdalmdimtranslate command line option, 79
 <src_datasource>
 gdal_translate command line option, 39
 gdaladdo command line option, 47
 <src_dsname>
 ogrmerge.py command line option, 95
 <srcfile>
 gdaladdo command line option, 51
 gdallocationinfo command line option, 55
 gdaltransform command line option, 41
 gdalwarp command line option, 20
 <srs_def>
 gdalsrsinfo command line option, 57
 <x>
 gdallocationinfo command line option, 55
 <y>
 gdallocationinfo command line option, 55

A

alg ZevenbergenThorne
 command line option, 28
 autoconnect <tolerance>
 gnmmmanage command line option, 98

C

change
 gnmmmanage command line option, 98
 color_text_file
 command line option, 30
 command line option
 -3d, 25
 -A <filename>, 65
 --A_band=<n>, 65
 --NoDataValue=<value>, 65
 --allBands=[A-Z], 65
 --calc=expression, 65
 --cflags, 68
 --co=<option>, 65
 --config <key> <value>, 5, 80
 --creation-option=<option>, 65
 --debug, 66
 --debug <value>, 5, 80
 --format <format>, 5, 80
 --format=<gdal_format>, 65
 --formats, 5, 68, 80
 --help, 65
 --help-general, 5, 63, 80, 92
 --libs, 68
 --ogr-enabled, 68
 --optfile <filename>, 5, 80
 --outfile=<filename>, 65
 --overwrite, 66
 --prefix, 68
 --quiet, 66
 --type=<datatype>, 65
 --version, 5, 68, 80
 -a <name>, 25
 -a_nodata <value>, 64
 -a_srs <srs_def>, 63
 -a_srs <srs>, 6
 -a_ullr ulx uly lrx lry:, 63
 -a_ulurll ulx uly urx ury llx lly:, 63
 -alpha, 30
 -alt <altitude>, 29
 -amax <name>, 25
 -amin <name>, 25
 -approx_stats, 63
 -az <azimuth>, 29
 -b <band>, 25, 28, 66
 -b band, 54
 -bitdepth <val>, 67

-co <NAME=VALUE>, 6, 28, 67
 -combined, 29
 -compute_edges, 28
 -create, 93
 -dsco <NAME=VALUE>, 25, 93
 -e <base>, 26
 -et <max_pixel_err>, 62
 -exact_color_entry, 30
 -f <format_name>, 92
 -f <format>, 25, 69
 -fl <level>, 26
 -gcp pixel line easting northing
 [elevation], 64
 -get_coord, 93
 -get_pos, 93
 -get_subline, 94
 -h, 65
 -i <interval>, 26
 -igor, 29
 -inodata, 25
 -l <src_line_datasource_name>, 93
 -lco <NAME=VALUE>, 26, 93
 -lf <field_name>, 93
 -ln <layer_name>, 93
 -m <position>, 93
 -mask filename, 54
 -mb <position>, 94
 -md max_distance, 54
 -me <position>, 94
 -mo META-TAG=VALUE, 64
 -multidirectional, 29
 -nearest_color_entry, 30
 -nlm <name>, 26
 -nodata <val>, 67
 -nomask, 54
 -o <dst_datasource_name>, 93
 -o name=value, 54
 -of <field_name>, 93
 -of <format>, 6, 28
 -of <format>:, 66
 -of format, 54
 -off <offset>, 26
 -offset <value>, 64
 -on <layer_name>, 93
 -oo NAME=VALUE, 64
 -p, 26, 29
 -p <src_repers_datasource_name>, 93
 -pf <field_name>, 93
 -pm <pos_field_name>, 93
 -pn <layer_name>, 93
 -progress, 92
 -q, 26, 28, 54, 67
 -quiet, 92
 -r, 69
 -r {nearest,bilinear,cubic
 (default),cubicspline,lanczos,average},
 67
 -r <src_parts_datasource_name>, 93
 -rn <layer_name>, 93
 -ro, 63
 -s <scale>, 29
 -s <step>, 93
 -s_srs <srs_defn>, 62
 -s_srs <srs>, 6
 -scale <value>, 64
 -setstatsmin max mean stddev, 63
 -si smooth_iterations, 54
 -snodata <value>, 25
 -spat_adjust {union(default),intersection,none},
 67
 -stats, 63
 -t_srs <srs_defn>, 62
 -t_srs <srs>, 6
 -threads {ALL_CPUS,number}, 67
 -tr <xres> <yres>, 63
 -trigonometric, 30
 -u, 69
 -units <value>, 64
 -unsetgt, 63
 -unsetmd, 64
 -unsetnodata, 64
 -unsetrpc, 63
 -unsetstats, 63
 -w <weight_val>, 66
 -x <long>, 93
 -y <lat>, 93
 -z <factor>, 29
 -zero_for_flat, 30
 <datasetname>, 69
 <mode>, 27, 69
 <newdatasetname>, 69
 <out_dataset>, 67
 <pan_dataset>, 67
 <spectral_dataset>[,band=num], 67
 alg ZevenbergenThorne, 28
 color_text_file, 30
 dstfile, 54
 input_dem, 28
 output_xxx_map, 28
 srcfile, 54
 connect <gfid_src> <gfid_tgt>
 <gfid_con>
 gnmmmanage command line option, 97
 create
 gnmmmanage command line option, 97
D
 delete

gnmmanage command line option, 98
 dijkstra <start_gfid> <end_gfid>
 gnmanalyse command line option, 98
 disconnect <gfid_src> <gfid_tgt>
 <gfid_con>
 gnmmanage command line option, 98
 dstfile
 command line option, 54

F

fields, 948

G

gdal2tiles, 35
 gdal_calc, 65
 gdal_contour, 25
 gdal_edit, 62
 gdal_fillnodata, 54
 gdal_grid, 45
 gdal_merge, 34
 gdal_pansharpen, 66
 gdal_polygonize, 52
 gdal_proximity, 51
 gdal_rasterize, 37
 gdal_retile, 44
 gdal_sieve, 53
 gdal_translate, 9
 gdal_translate command line option
 -3d, 38
 --bingkey=<BINGKEY>, 37
 --copyright=<COPYRIGHT>, 37
 --force-kml, 36
 --googlekey=<GOOGLEKEY>, 37
 --help, 36
 --no-kml, 36
 --processes=<NB_PROCESSES>, 36
 --profile=<PROFILE>, 35
 --quiet, 36
 --resampling=<RESAMPLING>, 36
 --resume, 36
 --s_srs=<SRS>, 36
 --srcnodata=<NODATA>, 36
 --tilesize=<TILESIZE>, 36
 --title=<TITLE>, 37
 --url=<URL>, 36
 --verbose, 36
 --version, 36
 --webviewer=<WEBVIEWER>, 37
 --xyz, 36
 --zoom=<ZOOM>, 36
 -a <NODATA>, 36
 -a <attribute_name>, 38
 -a_nodata <value>, 12, 38
 -a_offset<value>, 11

-a_scale <value>, 11
 -a_srs <srs_def>, 11, 38
 -a_ullr <ulx> <uly> <lrx> <lry>, 11
 -add, 38
 -at, 38
 -b <BINGKEY>, 37
 -b <band>, 10, 37
 -burn <value>, 38
 -c <COPYRIGHT>, 37
 -co <NAME=VALUE>, 12, 39, 44
 -colorinterp <red|green|blue|alpha|gray|undefined>
 12
 -colorinterp_X
 <red|green|blue|alpha|gray|undefined>,
 12
 -csv <csvFileName>, 45
 -csvDelim <column delimiter>, 45
 -dialect <dialect>, 38
 -e, 36
 -eco, 11
 -epo, 11
 -expand gray|rgb|rgba, 10
 -exponent <exp_val>, 11
 -g <GOOGLEKEY>, 37
 -gcp <pixel> <line> <easting>
 <northing> <elevation>, 12
 -h, 36
 -i, 37
 -init <value>, 38
 -k, 36
 -l <layername>, 38
 -levels <numberOfLevels>, 44
 -mask <band>, 10
 -mo META-TAG=VALUE, 12
 -n, 36
 -nogcp, 12
 -norat, 12
 -of <format>, 10, 38, 44
 -oo NAME=VALUE, 12
 -optim {[AUTO]/VECTOR/RASTER}}, 39
 -ot <type>, 10, 39, 44
 -outsize <xsize>[%]|0 <ysize>[%]|0,
 10
 -overlap< <val_in_pixel>, 44
 -p <PROFILE>, 35
 -projwin <ulx> <uly> <lrx> <lry>, 11
 -projwin_srs <srs_def>, 11
 -ps <pixelsize_x> <pixelsize_y>, 44
 -pyramidOnly, 44
 -q, 12, 36, 39
 -r {nearest (default), bilinear, cubic, cubicsplin
 10
 -r <RESAMPLING>, 36
 -r <algorithm>, 45

```

-resume, 45
-s <SRS>, 36
-s_srs <srs_def>, 45
-scale [src_min src_max [dst_min
    dst_max]], 10
-sds, 12
-sql <select_statement>, 38
-srcwin <xoff> <yoff> <xsize>
    <ysize>, 11
-stats, 12
-strict, 10
-t <TITLE>, 37
-tap, 39
-targetDir <directory>, 44
-te <xmin> <ymin> <xmax> <ymax>, 39
-tileIndex <tileIndexName>, 45
-tileIndexField
    <tileIndexFieldName>, 45
-to NAME=VALUE, 38
-tr <xres> <yres>, 10, 39
-ts <width> <height>, 39
-u <URL>, 36
-unscale, 11
-useDirForEachRow, 45
-v, 36, 44
-w <WEBVIEWER>, 37
-where <expression>, 38
-z <ZOOM>, 36
<dst_dataset>, 12
<dst_filename>, 39
<src_dataset>, 12
<src_datasource>, 39
gdal_viewshed, 71
gdal_viewshed command line option
    -a_nodata <value>, 71
    -b <band>, 71
    -cc <value>, 72
    -co <NAME=VALUE>, 71
    -iv <value>, 72
    -md <value>, 72
    -om <output mode>, 72
    -ov <value>, 72
    -ox <value>, 71
    -oy <value>, 72
    -oz <value>, 72
    -tz <value>, 72
    -vv <value>, 72
gdal-config, 68
gdaladdo, 13
gdaladdo command line option
    -8, 52
    -a <[algorithm[:parameter1=value1]][:parameter2=value2]...>, 46
    -a_nodata <output_nodata_value>, 34
    -a_srs <srs_def>, 46
    -b <band>, 13, 33, 53
    -clean, 14
    -clipsrc [xmin ymin xmax
        ymax] | WKT | datasource | spat_extent,
        46
    -clipsrclayer <layername>, 47
    -clipsrcsql <sql_statement>, 46
    -clipsrcwhere <expression>, 47
    -co <NAME=VALUE>, 34, 47, 51
    -createonly, 35
    -distunits PIXEL|GEO, 52
    -dstband <n>, 51
    -f <ogr_format>, 53
    -fixed-buf-val <n>, 52
    -init <"value(s)">, 35
    -l <layername>, 47
    -mask <filename>, 52
    -maxdist <n>, 52
    -minsize <val>, 14
    -n <color>, 32
    -n <nodata_value>, 34
    -nodata <n>, 52
    -nomask, 52
    -o <out_filename>, 34
    -of <format>, 32-34, 46, 51
    -oo NAME=VALUE, 14
    -ot <type>, 34, 46, 52
    -outsize <xsize ysize>, 46
    -pct, 34
    -pct <palette_file>, 32
    -ps <pixelsize_x> <pixelsize_y>, 34
    -q, 47, 53
    -r {nearest (default), average, gauss, cubic, cubic
        13
    -rgba, 33
    -ro, 13
    -separate, 34
    -spat <xmin> <ymin> <xmax> <ymax>,
        46
    -sql <select_statement>, 47
    -srcband <n>, 51
    -tap, 34
    -txe <xmin> <xmax>, 46
    -tye <ymin> <ymax>, 46
    -ul_lr <ulx> <uly> <lrx> <lry>, 34
    -use_input_nodata YES/NO, 52
    -v, 34
    -values <n>, <n>, <n>, 52
    -where <expression>, 47
    -z_increase <increase_value>, 46
    -z_multiply <multiply_value>, 46
    -zfield <field_name>, 46
    <dest_file>, 32, 33

```

```

<dst_filename>, 47
<dstfile>, 51
<fieldname>, 53
<filename>, 14
<layer>, 53
<levels>, 14
<out_file>, 53
<raster_file>, 53
<source_file>, 32, 33
<src_datasource>, 47
<srcfile>, 51
gdalbuildvrt, 22
gdalbuildvrt command line option
    -a_srs <srs_def>, 24
    -addalpha, 23
    -allow_projection_difference, 24
    -b <band>, 23
    -hidenodata, 23
    -input_file_list <mylist.txt>, 24
    -oo NAME=VALUE, 24
    -optim {[AUTO]/VECTOR/RASTER}}, 24
    -overwrite, 24
    -q, 24
    -r {nearest (default), bilinear, cubic, cubic_splines,
        24
        lanczos, average, mode},
    -resolution {highest|lowest|average|used},
        23
    -sd< <subdataset>, 23
    -separate, 24
    -srcnodata <value> [<value>...], 23
    -tap, 23
    -te xmin ymin xmax ymax, 23
    -tileindex, 23
    -tr <res> <yres>, 23
    -vrtnodata <value> [<value>...], 24
gdalcompare.py, 70
gdaldem, 26
gdalinfo, 7
gdalinfo command line option
    -approx_stats, 7
    -array {array_name}, 75
    -arrayoption <NAME=VALUE>, 75
    -checksum, 7
    -detailed, 75
    -hist, 7
    -json, 7
    -limit {number}, 75
    -listmdd, 8
    -mdd <domain>|all, 8
    -mm, 7
    -noct, 7
    -nofl, 8
    -nogcp, 7
    -nomd, 7
    -nopretty, 75
    -norat, 7
    -oo <NAME=VALUE>, 8, 75
    -proj4, 8
    -sd <n>, 8
    -sds, 70
    -stats, 7
    -wkt_format WKT1|WKT2|WKT2_2015|WKT2_2018,
        8
    <golden_file>, 70
    <new_file>, 70
gdallocationinfo, 54
gdallocationinfo command line option
    -b <band>, 55
    -geoloc, 55
    -l_srs <srs_def>, 55
    -lifonly, 55
    -oo NAME=VALUE, 55
    -overview <overview_level>, 55
    -valonly, 55
    -wgs84, 55
    -xml, 55
    <srcfile>, 55
    <res>, 55
    <y>, 55
gdalmanage, 69
gdalmdiminfo, 75
gdalmdimtranslate, 78
gdalmdimtranslate command line option
    -array <array_spec>, 78
    -co <NAME=VALUE>, 78
    -group <group_spec>, 78
    -of <format>, 78
    -scaleaxes <scaleaxes_spec>, 79
    -subset <subset_spec>, 79
    <dst_dataset>, 79
    <src_dataset>, 79
gdalmove, 62
gdalsrsinfo, 57
gdalsrsinfo command line option
    -V, 57
    --single-line, 57
    -e, 57
    -o <out_type>, 57
    <srs_def>, 57
gdaltindex, 21
gdaltransform, 40
gdaltransform command line option
    -ct <string>, 40
    -gcp <pixel> <line> <easting>
        <northing> [<elevation>], 41
    -geoloc, 41
    -i, 41
    -order <n>, 41

```

```

-output_xy, 41
-rpc, 41
-s_srs <srs_def>, 40
-t_srs <srs_def>, 40
-to NAME=VALUE, 41
-tps, 41
<dstfile>, 41
<srcfile>, 41
gdalwarp, 16
gdalwarp command line option
-cblend <distance>, 19
-cl <layername>, 19
-co <NAME=VALUE>, 19
-crop_to_cutline, 19
-csql <query>, 19
-ct <string>, 16
-cutline <datasource>, 19
-cvmd <meta_conflict_value>, 19
-cwhere <expression>, 19
-doo <NAME=VALUE>, 19
-dstalpa, 18
-dstnodata <value [value...]>, 18
-et <err_threshold>, 17
-geoloc, 17
-multi, 19
-nomd, 19
-nosrcalpha, 18
-novshiftgrid, 16
-of <format>, 19
-oo <NAME=VALUE>, 19
-order <n>, 17
-ot <type>, 18
-overwrite, 19
-ovr <level|AUTO|AUTO-n|NONE>, 17
-q, 19
-r <resampling_method>, 18
-refine_gcps <tolerance
    minimum_gcps>, 17
-rpc, 17
-s_srs <srs def>, 16
-setci, 19
-srcalpha, 18
-srcnodata <value [value...]>, 18
-t_srs <srs_def>, 16
-tap, 17
-te <xmin ymin xmax ymax>, 17
-te_srs <srs_def>, 17
-to <NAME=VALUE>, 16
-tps, 17
-tr <xres> <yres>, 17
-ts <width> <height>, 17
-wm <memory_in_mb>, 18
-wo ` "NAME=VALUE"`, 17
-wt <type>, 18
    <dstfile>, 20
    <srcfile>, 20
gnmanalyse, 98
gnmanalyse command line option
-alo NAME=VALUE, 99
-d <ds_name>, 99
-dsco NAME=VALUE, 99
-f <ds_format>, 99
-l <layer_name>, 99
-lco NAME=VALUE, 99
<gnm_name>, 99
dijkstra <start_gfid> <end_gfid>, 98
kpaths <start_gfid> <end_gfid>, 98
resource, 98
gnmmanage, 97
gnmmanage command line option
-bl <gfid>, 98
-c <cost>, 97
-dir <dir>, 97
-dsco NAME=VALUE, 97
-f <format_name>, 97
-ic <invcost>, 97
-info, 97
-l layer_name, 97
-t_srs <srs_name>, 97
-unbl <gfid>, 98
-unblall, 98
<gnm_name>, 98
<layer>, 98
autoconnect <tolerance>, 98
change, 98
connect <gfid_src> <gfid_tgt>
    <gfid_con>, 97
create, 97
delete, 98
disconnect <gfid_src> <gfid_tgt>
    <gfid_con>, 98
import <src_dataset_name>, 97
rule <rule_str>, 98

I
import <src_dataset_name>
    gnmmanage command line option, 97
index_file
    ogrtindex command line option, 21
input_dem
    command line option, 28

K
kpaths <start_gfid> <end_gfid>
    gnmanalyse command line option, 98

N
nearblack, 42

```

O

ogr2ogr, 84

ogr2ogr command line option

- a_srs <srs_def>, 86
- addfields, 89
- append, 85
- clipdst <xmin> <ymin> <xmax> <ymax>, 87
- clipdstlayer <layername>, 87
- clipdstsql <sql_statement>, 87
- clipdstwhere <expression>, 87
- clipsrc [xmin ymin xmax ymax] | WKT | datasource | spat_extent, 87
- clipsrclayer <layername>, 87
- clipsrcsql <sql_statement>, 87
- clipsrcwhere <expression>, 87
- ct <string>, 86
- datelineoffset, 87
- dialect <dialect>, 85
- dim <val>, 86
- doo NAME=VALUE, 87
- ds_transaction, 87
- dsco NAME=VALUE, 86
- explodecollections, 88
- f <format_name>, 85
- fid fid, 87
- fieldTypeToString type1,..., 88
- fieldmap, 88
- forceNullable, 89
- gcp <ungeoref_x> <ungeoref_y> <georef_x> <georef_y> <elevation>, 88
- geomfield <field>, 86
- gt n, 87
- lco NAME=VALUE, 86
- limit nb_features, 87
- makevalid, 88
- mapFieldType srctype|All=dsttype,..., 88
- maxsubfields <val>, 88
- mo META-TAG=VALUE, 89
- nln <name>, 86
- nlt <type>, 86
- noNativeData, 89
- nomd, 89
- oo NAME=VALUE, 87
- order <n>, 88
- overwrite, 85
- preserve_fid, 87
- progress, 85
- relaxedFieldNameMatch, 89
- s_srs <srs_def>, 86
- segmentize <max_dist>, 88

- select <field_list>, 85
- simplify <tolerance>, 88
- skipfailures, 85
- spat <xmin> <ymin> <xmax> <ymax>, 85
- spat_srs <srs_def>, 86
- splitlistfields, 88
- sql <sql_statement>, 85
- t_srs <srs_def>, 86
- tps, 88
- unsetDefault, 89
- unsetFid, 89
- unsetFieldWidth, 88
- update, 85
- where restricted_where, 85
- wrapdateline, 87
- zfield <field_name>, 88

ogrinfo, 80

ogrinfo command line option

- formats, 82
- al, 81
- dialect <dialect>, 81
- fid <fid>, 81
- fields YES|NO:, 81
- geom YES|NO|SUMMARY|WKT|ISO_WKT, 81
- geomfield <field>, 81
- listmdd, 81
- mdd <domain>, 82
- nocount, 82
- noextent, 82
- nogeomtype, 82
- nomd, 81
- oo NAME=VALUE, 81
- q, 81
- rl, 81
- ro, 81
- so, 81
- spat <xmin> <ymin> <xmax> <ymax>, 81
- sql <statement>, 81
- where <restricted_where>, 81
- wkt_format <format>, 82
- <datasource_name>, 82
- <layer>, 82

ogrlineref, 92

ogrmerge, 94

ogrmerge.py command line option

- a_srs <srs_def>, 95
- append, 95
- dsco <NAME=VALUE>, 95
- f <format>, 95
- field_strategy FirstLayer|Union|Intersection, 96
- lco <NAME=VALUE>, 95

-nln <layer_name_template>, 95
 -o <out_dsname>, 94
 -overwrite_ds, 95
 -overwrite_layer, 95
 -progress, 96
 -s_srs <srs_def>, 95
 -single, 95
 -skipfailures, 96
 -src_geom_type
 <geom_type_name[,geom_type_name]*]>,
 95
 -src_layer_field_content
 <layer_name_template>, 96
 -src_layer_field_name <name>, 96
 -t_srs <srs_def>, 95
 -update, 95
 <src_dsname>, 95
 ogrtindex, 91
 ogrtindex command line option
 -accept_different_schemas, 91
 -co `"NAME=VALUE"`, 42
 -color <c1,c2,c3...cn>, 43
 -f <format>, 21
 -f <output_format>, 91
 -lname <name>, 91
 -lnum <n>, 91
 -lyr_name <name>, 21
 -nb <non_black_pixels>, 43
 -near <dist>, 43
 -o <outfile>, 42
 -of <format>, 42
 -q, 43
 -setalpha, 43
 -setmask, 43
 -skip_different_projection, 21, 91
 -src_srs_format <format>, 91
 -src_srs_format <type>, 21
 -src_srs_name <field_name>, 21, 91
 -t_srs <target_srs>, 91
 -t_srs <target_srs>:, 21
 -tileindex <field_name>, 21, 91
 -white, 43
 -write_absolute_path, 21, 91
 <gdal_file>, 21
 <infile>, 43
 index_file, 21
 output_xxx_map
 command line option, 28

P

pct2rgb, 33

R

resource

gnmanalyse command line option, 98
 rgb2pct, 32
 rule <rule_str>
 gnmmmanage command line option, 98

S

srcfile
 command line option, 54