GDAL/OGR user docs

GDAL/OGR user docs

GDAL/OGR user docs

GDAL Utilities

The following utility programs are distributed with GDAL.

- gdalinfo - report information about a file.
- gdal_translate - Copy a raster file, with control of output format.
- gdaladdo - Add overviews to a file.
- gdalwarp - Warp an image into a new coordinate system.
- gdaltindex - Build a MapServer raster tileindex.
- gdalbuildvrt - Build a VRT from a list of datasets.
- gdal_contour - Contours from DEM.
- gdaldem - Tools to analyze and visualize DEMs.
- rgb2pct.py - Convert a 24bit RGB image to 8bit paletted.
- pct2rgb.py - Convert an 8bit paletted image to 24bit RGB.
- gdal_merge.py - Build a quick mosaic from a set of images.
- gdal2tiles.py - Create a TMS tile structure, KML and simple web viewer.
- gdal_rasterize - Rasterize vectors into raster file.
- gdaltransform - Transform coordinates.
- nearblack - Convert nearly black/white borders to exact value.
- gdal_retile.py - Retiles a set of tiles and/or build tiled pyramid levels.
- gdal_grid - Create raster from the scattered data.
- gdal_proximity.py - Compute a raster proximity map.
- gdal_polygonize.py - Generate polygons from raster.
- gdal_sieve.py - Raster Sieve filter.
- gdal_fillnodata.py - Interpolate in nodata regions.
- gdallocationinfo - Query raster at a location.
- gdal-config - Get options required to build software using GDAL.

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 commandline. 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.

  Many utilities default to creating GeoTIFF files if a format is not specified. File extensions are not used to guess output format, nor are extensions generally added by GDAL if not indicated in the filename by the user.

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

  The creation options available vary by format driver, and some simple formats have no creation options at all. A list of options supported for a format can be listed with the "--format <format>" commandline option but the web page for the format is the definitive source of information on driver creation options.

-a_srs *SRS*
: Several utilities, (gdal_translate and gdalwarp) include the ability to specify coordinate systems with commandline 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 Definitions*: A PROJ.4 definition string can be used as a coordinate system. For instance "+proj=utm +zone=11 +datum=WGS84". Take care to keep the proj.4 string together as a single argument to the command (usually by double quoting).
- ◊ *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 commandline 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 to 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*: The name of a file containing WKT, PROJ.4 strings, or XML/GML coordinate system definitions can be provided.

General Command Line Switches

All GDAL command line utility programs support the following "general" 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 (ie. supports CreateCopy); 'rw+' is read, write and update (ie. supports Create). A 'v' is appended for formats supporting virtual IO (/vsimem, /vsigzip, /vsizip, etc). 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 *file*
: Read the named file and substitute the contents into the commandline 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 commandline options and exit.

gdalinfo

lists information about a raster dataset

SYNOPSIS

```
gdalinfo [--help-general] [-mm] [-stats] [-hist] [-nogcp] [-nomd]
         [-noct] [-nofl] [-checksum] [-mdd domain]*
         [-sd subdataset] datasetname
```

DESCRIPTION

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

-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.

-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.

-noct
: Suppress printing of color table.

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

-mdd domain
: Report metadata for the specified domain

-nofl
: (GDAL >= 1.9.0) Only display the first file of the file list.

-sd *subdataset*
: (GDAL >= 1.9.0) If the input dataset contains several subdatasets read and display a subdataset with specified number (starting from 1). This is an alternative of giving the full subdataset name.

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.

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
```

gdal_translate

converts raster data between different formats

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[%] ysize[%]]
       [-unscale] [-scale [src_min src_max [dst_min dst_max]]]
       [-srcwin xoff yoff xsize ysize] [-projwin ulx uly lrx lry]
       [-a_srs srs_def] [-a_ullr ulx uly lrx lry] [-a_nodata value]
       [-gcp pixel line easting northing [elevation]]*
       [-mo "META-TAG=VALUE"]* [-q] [-sds]
       [-co "NAME=VALUE"]* [-stats]
       src_dataset dst_dataset
```

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*
: For the output bands to be of the indicated data type.

-strict:
: Do'nt be forgiving of mismatches and lost data when translating to the output format.

-of *format*:
: Select the output format. The default is GeoTIFF (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. Starting with GDAL 1.8.0, *band* can also be set to "mask,1" (or just "mask") to mean the mask band of the 1st band of the input dataset.

-mask *band*:
: (GDAL >= 1.8.0) 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 explicitely used to ben 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*:
: (From GDAL 1.6.0) To expose a dataset with 1 band with a color table as a dataset with 3 (RGB) or 4 (RGBA) bands. Usefull for output drivers such as JPEG, JPEG2000, MrSID, ECW that don't support color indexed datasets. The 'gray' value (from GDAL 1.7.0) enables to expand a dataset with a color table that only contains gray levels to a gray indexed dataset.

-outsize *xsize[%] ysize[%]*:
: 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.

-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.

-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.

-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.

-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.

-a_nodata *value*:
: Assign a specified nodata value to output bands. Starting with GDAL 1.8.0, can be set to *none* to avoid setting a nodata value to the output file if one exists for the source file

-mo *"META-TAG=VALUE"*:
: Passes a metadata key and value to set on the output dataset if possible.

-co *"NAME=VALUE"*:
: Passes a creation option to the output format driver. Multiple **-co** options may be listed. See format specific documentation for legal creation options for each format.

-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 or OGDI that have subdatasets.

-stats:
: (GDAL >= 1.8.0) Force (re)computation of statistics.

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.

EXAMPLE

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

Starting with GDAL 1.8.0, 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
```

Starting with GDAL 1.8.0, 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
```

gdaladdo

builds or rebuilds overview images

SYNOPSIS

```
gdaladdo [-r {nearest,average,gauss,cubic,average_mp,average_magphase,mode}]
         [-ro] [-clean] [--help-general] filename levels
```

DESCRIPTION

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

-r *{nearest (default),average,gauss,cubic,average_mp,average_magphase,mode}*:
: Select a resampling algorithm.

-ro:
: (available from GDAL 1.6.0) open the dataset in read-only mode, in order to generate external overview (for GeoTIFF especially).

-clean:
: (available from GDAL 1.7.0) remove all overviews.

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.

Mode (available from GDAL 1.6.0) selects the value which appears most often of all the sampled points. *average_mp* is unsuitable for use. *Average_magphase* averages complex data in mag/phase space. *Nearest* and *average* are applicable to normal image data. *Nearest* applies a nearest neighbour (simple sampling) resampler, while *average* computes the average of all non-NODATA contributing pixels. *Cubic* resampling (available from GDAL 1.7.0) applies a 4x4 approximate cubic convolution kernel. *Gauss* resampling (available from GDAL 1.6.0) 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.

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 independantly 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.

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. (eg --config USE_RRD YES)

External overviews in GeoTIFF format

External overviews created in TIFF format may be compressed using the COMPRESS_OVERVIEW configuration option. All compression methods, supported by the GeoTIFF driver, are available here. (eg --config COMPRESS_OVERVIEW DEFLATE). The photometric interpretation can be set with --config PHOTOMETRIC_OVERVIEW {RGB,YCBCR,...}, and the interleaving with --config INTERLEAVE_OVERVIEW {PIXEL|BAND}.

For JPEG compressed external overviews, the JPEG quality can be set with "--config JPEG_QUALITY_OVERVIEW value" (GDAL 1.7.0 or later).

For LZW or DEFLATE compressed external overviews, the predictor value can be set with "--config PREDICTOR_OVERVIEW 1|2|3" (GDAL 1.8.0 or later).

To produce the smallest possible JPEG-In-TIFF overviews, you should use :

```
--config COMPRESS_OVERVIEW JPEG --config PHOTOMETRIC_OVERVIEW YCBCR --config INTERLEAVE_OVERVIEW PI
```

Starting with GDAL 1.7.0, 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_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 GeoTIFF driver for further explanations on all those options.

EXAMPLE

Example:

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
```

gdalwarp

image reprojection and warping utility

SYNOPSIS

Usage:

```
gdalwarp [--help-general] [--formats]
    [-s_srs srs_def] [-t_srs srs_def] [-to "NAME=VALUE"]
    [-order n | -tps | -rpc | -geoloc] [-et err_threshold]
    [-refine_gcps tolerance [minimum_gcps]]
    [-te xmin ymin xmax ymax] [-tr xres yres] [-tap] [-ts width height]
    [-wo "NAME=VALUE"] [-ot Byte/Int16/...] [-wt Byte/Int16]
    [-srcnodata "value [value...]"] [-dstnodata "value [value...]"] -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]
    srcfile* dstfile
```

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*:
: source spatial reference set. The coordinate systems that can be passed are anything supported by the OGRSpatialReference.SetFromUserInput() call, which includes EPSG PCS and GCSes (ie. EPSG:4296), PROJ.4 declarations (as above), or the name of a .prf file containing well known text.

-t_srs *srs_def*:
: target spatial reference set. The coordinate systems that can be passed are anything supported by the OGRSpatialReference.SetFromUserInput() call, which includes EPSG PCS and GCSes (ie. EPSG:4296), PROJ.4 declarations (as above), or the name of a .prf file containing well known text.

-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.

-et *err_threshold*:
: error threshold for transformation approximation (in pixel units - defaults to 0.125).

-refine_gcps *tolerance minimum_gcps*:
: (GDAL >= 1.9.0) 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).

-tr *xres yres*:
: set output file resolution (in target georeferenced units)

-tap:
: (GDAL >= 1.8.0) (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

-wo *"NAME=VALUE"*:
: Set a warp options. The GDALWarpOptions::papszWarpOptions docs show all options. Multiple **-wo** options may be listed.

-ot *type*:
: For the output bands to be of the indicated data type.

-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.

-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.

-dstalpha:
: Create an output alpha band to identify nodata (unset/transparent) pixels.

-wm *memory_in_mb*:
: Set the amount of memory (in megabytes) that the warp API is allowed to use for caching.

-multi:
: Use multithreaded warping implementation. Multiple threads will be used to process chunks of image and perform input/output operation simultaneously.

-q:
: Be quiet.

-of *format*:
: Select the output format. The default is GeoTIFF (GTiff). Use the short format name.

-co *"NAME=VALUE"*:
: passes a creation option to the output format driver. Multiple **-co** options may be listed. See 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:
: (GDAL >= 1.8.0) Crop the extent of the target dataset to the extent of the cutline.

-overwrite:
: (GDAL >= 1.8.0) Overwrite the target dataset if it already exists.

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 accomodate 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 georeferenced units of the destination file. When outputing 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.

EXAMPLE

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' raw_spot.tif utm11.tif
```

For instance, the second channel of an ASTER image stored in HDF with control points mapping the corners to lat/long could be warped to a UTM projection with a command like this:

```
gdalwarp HDF4_SDS:ASTER_L1B:"pg-PR1B0000-2002031402_100_001":2
   pg-PR1B0000-2002031402_100_001_2.tif
```

gdaltindex

builds a shapefile as a raster tileindex

SYNOPSIS

```
gdaltindex [-tileindex field_name] [-write_absolute_path] [-skip_different_projection]
   index_file [gdal_file]*
```

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.

- The shapefile (index_file) will be created if it doesn't already exist, otherwise it will append to the existing file.
- The default tile index field is 'location'.
- Raster filenames will be put in the file exactly as they are specified on the commandline unless the option -write_absolute_path is used.
- If -skip_different_projection is specified, only files with same projection ref as files already inserted in the tileindex will be inserted.
- Simple rectangular polygons are generated in the same coordinate system as the rasters.

EXAMPLE

Example:

```
gdaltindex doq_index.shp doq/*.tif
```

gdalbuildvrt

Builds a VRT from a list of datasets. (compiled by default since GDAL 1.6.1)

SYNOPSIS

```
gdalbuildvrt [-tileindex field_name] [-resolution {highest|lowest|average|user}]
             [-tr xres yres] [-tap] [-separate] [-allow_projection_difference] [-q]
             [-te xmin ymin xmax ymax] [-addalpha] [-hidenodata]
             [-srcnodata "value [value...]"] [-vrtnodata "value [value...]"]
             [-input_file_list my_liste.txt] [-overwrite] output.vrt [gdalfile]*
```

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 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 *stacked* band in the VRT band. 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 may depend on the order they are inserted in the VRT file, but this behaviour should not be relied on.

This utility is somehow equivalent to the gdal_vrtmerge.py utility and is build by default in GDAL 1.6.1.

-tileindex:
: Use the specified value as the tile index field, instead of the default value with 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. 'average' is the default. 'highest' will pick the smallest values of pixel dimensions within the set of source rasters. 'lowest' will pick the largest values of pixel dimensions within the set of source rasters. 'average' will compute an average of pixel dimensions within the set of source rasters. 'user' is new in GDAL 1.7.0 and must be used in combination with the -tr option to specify the target resolution.

-tr xres yres :
: (starting with GDAL 1.7.0) 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:
: (GDAL >= 1.8.0) (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 :
: (starting with GDAL 1.7.0) 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:
: (starting with GDAL 1.7.0) Adds an alpha mask band to the VRT when the source raster have none. Mainly useful for RGB sources (or grey-level sources). The alpha band is filled on-the-fly with the value 0 in areas without any source raster, and with value 255 in areas with source raster. The effect is that a RGBA viewer will render the areas without source rasters as transparent and areas with source rasters as opaque. This option is not compatible with -separate.

-hidenodata:
: (starting with GDAL 1.7.0) 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...]*:
: (starting with GDAL 1.7.0) 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 instrinsic 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.

-vrtnodata *value [value...]*:
: (starting with GDAL 1.7.0) 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, instrinsic 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:
: (starting with GDAL 1.7.0) Place each input file into a separate *stacked* 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:
: (starting with GDAL 1.7.0) 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.

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

-q:
: (starting with GDAL 1.7.0) To disable the progress bar on the console

-overwrite:
: Overwrite the VRT if it already exists.

EXAMPLE

Example:

```
gdalbuildvrt doq_index.vrt doq/*.tif
gdalbuildvrt -input_file_list my_liste.txt doq_index.vrt
gdalbuildvrt -separate rgb.vrt red.tif green.tif blue.tif
gdalbuildvrt -hidenodata -vrtnodata "0 0 255" doq_index.vrt doq/*.tif
```

gdal_contour

builds vector contour lines from a raster elevation model

SYNOPSIS

```
Usage: gdal_contour [-b <band>] [-a <attribute_name>] [-3d] [-inodata]
                    [-snodata n] [-f <formatname>] [-i <interval>]
                    [-off <offset>] [-fl <level> <level>...]
                    [-nln <outlayername>]
                    <src_filename> <dst_filename>
```

DESCRIPTION

This program generates a vector contour file from the input raster elevation model (DEM).

Starting from version 1.7 the contour line-strings will be oriented consistently. 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.

-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, default is shapefiles.

-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.

-nln *outlayername*:
: Provide a name for the output vector layer. Defaults to "contour".

EXAMPLE

This would create 10meter 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
```

gdaldem

Tools to analyze and visualize DEMs. (since GDAL 1.7.0)

SYNOPSIS

Usage:

- To 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]
            [-compute_edges] [-b Band (default=1)] [-of format] [-co "NAME=VALUE"]* [-q]
```

- To 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]
```

- To 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]
```

- To 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" - To 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]
```

- To 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]
```

- To 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]
```

Notes : Scale is the ratio of vertical units to horizontal for Feet:Latlong use scale=370400, for Meters:LatLong use scale=111120) This utility has 7 different modes :

hillshade
: to generate a shaded relief map from any GDAL-supported elevation raster

slope
: to generate a slope map from any GDAL-supported elevation raster

aspect
: to generate an aspect map from any GDAL-supported elevation raster

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

TRI
: to generate a map of Terrain Ruggedness Index from any GDAL-supported elevation raster

TPI
: to generate a map of Topographic Position Index from any GDAL-supported elevation raster

roughness
: to 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. The default is GeoTIFF (GTiff). Use the short format name.

-compute_edges:
: (GDAL >= 1.8.0) Do the computation at raster edges and near nodata values

-alg *ZevenbergenThorne*:
: (GDAL >= 1.8.0) Use Zevenbergen & Thorne formula, instead of Horn's formula, to compute slope & aspect. The litterature 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"*:
: Passes a creation option to the output format driver. Multiple **-co** options may be listed. See 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. From GDAL 1.8.0, if -compute_edges is specified, gdaldem will compute values at image edges or if a nodata value is found in the 3x3 window, by interpolating missing values.

Modes

hillshade

This command outputs an 8-bit raster with a nice shaded relief effect. Its 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 *zFactor*:

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.

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 *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)

aspect

This command outputs a 32-bit float raster with values between 0 degree and 360 degree representing the azimuth that slopes are facing. The definition of the azimuth is such that : 0 degree means that the slope is facing the North, 90 degree it's facing the East, 180 degree it's facing the South and 270 degree it's facing the West (provided that the top of your input raster is north oriented). The aspect value -9999 is used as the nodata value to indicate undefined aspect in flat areas with slope=0.

The following specifics options are available :

-trigonometric:
: return trigonometric angle instead of azimuth. Thus 0 degree means East, 90 degree North, 180 degree West, 270 degree South

-zero_for_flat:
: return 0 for flat areas with slope=0, instead of -9999

By using those 2 options, the aspect returned by gdaldem aspect should be identical to the one of GRASS r.slope.aspect. Otherwise, it's identical to the one of Matthew Perry's aspect.cpp utility.

color-relief

This command outputs a 3-band (RGB) or 4-band (RGBA) raster with values are computed from the elevation and a text-based color configuration file, containing the association between various elevation values and the corresponding wished color. By default, the colors between the given elevation values are blended smoothly and the result is a nice colorized DEM. The -exact_color_entry or -nearest_color_entry options can be used to avoid that linear interpolation for values that don't match an index of the color configuration file.

The following specifics options are available :

color_text_file:
: text-based color configuration file

-alpha :
: add an alpha channel to the output raster

-exact_color_entry :
: use strict matching when searching in the color configuration file. If none matching color entry is found, the "0,0,0,0" RGBA quadruplet will be used

-nearest_color_entry :
: use the RGBA quadruplet corresponding to the closest entry in the color configuration file.

The color-relief mode is the only mode that supports VRT as output format. In that case, it will translate the color configuration file into appropriate <LUT> elements. Note that elevations specified as percentage will be translated as absolute values, which must be taken into account when the statistics of the source raster differ from the one that was used when building the VRT.

The text-based color configuration file generally contains 4 columns per line : the elevation value and the corresponding Red, Green, Blue component (between 0 and 255). The elevation value can be any floating point value, or the *nv* keyword for the nodata value.. The elevation can also be expressed as a percentage : 0% being the minimum value found in the raster, 100% the maximum value.

An extra column can be optionnaly 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.

Since GDAL 1.8.0, 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 tablulations and commma 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
```

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 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.

AUTHORS

Matthew Perry <perrygeo@gmail.com>, Even Rouault <even.rouault@mines-paris.org>, Howard Butler <hobu.inc@gmail.com>, Chris Yesson <chris.yesson@ioz.ac.uk>

Derived from code by Michael Shapiro, Olga Waupotitsch, Marjorie Larson, Jim Westervelt : U.S. Army CERL, 1993. GRASS 4.1 Reference Manual. U.S. Army Corps of Engineers, Construction Engineering Research Laboratories, Champaign, Illinois, 1-425.

See also

Documentation of related GRASS utilities :

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

rgb2pct.py

Convert a 24bit RGB image to 8bit paletted

SYNOPSIS

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

DESCRIPTION

This utility will compute an optimal pseudo-color table for a given RGB image using a median cut algorithm on a downsampled RGB histogram. Then it converts the image into a pseudo-colored image using the color table. This conversion utilizes Floyd-Steinberg dithering (error diffusion) to maximize output image visual quality.

-n *colors*:
: 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 consistant color table for multiple files. The *palette_file* must be a raster file in a GDAL supported format with a palette.

-of *format*:
: Format to generated (defaults to GeoTIFF). Same semantics as the **-of** flag for gdal_translate. Only output formats supporting pseudocolor 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.

EXAMPLE

If it is desired to hand create the palette, likely the simpliest 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>
```

pct2rgb.py

Convert an 8bit paletted image to 24bit RGB

SYNOPSIS

Usage:

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

DESCRIPTION

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

-of *format*:
    Format to generated (defaults to GeoTIFF).

-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 new '-expand rgb|rgba' option of gdal_translate obsoletes that utility.

gdal_merge.py

mosaics a set of images

SYNOPSIS

```
gdal_merge.py [-o out_filename] [-of out_format] [-co NAME=VALUE]*
              [-ps pixelsize_x pixelsize_y] [-tap] [-separate] [-v] [-pct]
              [-ul_lr ulx uly lrx lry] [-n nodata_value] [-init "value [value...]"]
              [-ot datatype] [-createonly] input_files
```

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*:
: Output format, defaults to GeoTIFF (GTiff).

-co *NAME=VALUE*:
: Creation option for output file. Multiple options can be specified.

-ot *datatype*:
: Force the output image bands to have a specific type. Use type names (ie. 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:
: (GDAL >= 1.8.0) (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 *stacked* band.

-pct:
: Grab a pseudocolor table from the first input image, and use it for the output. Merging pseudocolored 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*:
: (GDAL >= 1.9.0) 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.

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 initalized to 255.

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

gdal2tiles.py

generates directory with TMS tiles, KMLs and simple web viewers

SYNOPSIS

```
gdal2tiles.py [-title "Title"] [-publishurl http://yourserver/dir/]
              [-nogooglemaps] [-noopenlayers] [-nokml]
              [-googlemapskey KEY] [-forcekml] [-v]
              input_file [output_dir]
```

DESCRIPTION

This utility generates a directory with small tiles and metadata, following OSGeo Tile Map Service Specification. Simple web pages with viewers based on Google Maps and OpenLayers 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 webbrowser. You only need to upload generated directory into a web server.

GDAL2Tiles creates also necessary metadata for Google Earth (KML SuperOverlay), in case the supplied map uses EPSG:4326 projection.

World files and embedded georeference is used during tile generation, but you can publish a picture without proper georeference too.

-title *"Title"*:
: Title used for generated metadata, web viewers and KML files.

-publishurl *http://yourserver/dir/*:
: Address of a directory into which you are going to upload the result. It should end with slash.

-nogooglemaps:
: Do not generate Google Maps based html page.

-noopenlayers:
: Do not generate OpenLayers based html page.

-nokml:
: Do not generate KML files for Google Earth.

-googlemapskey *KEY*:
: Key for your domain generated on Google Maps API web page (http://www.google.com/apis/maps/signup.html).

-forcekml
: Force generating of KML files. Input file must use EPSG:4326 coordinates!

-v
: Generate verbose output of tile generation.

NOTE: gdal2tiles.py is a Python script that needs to be run against "new generation" Python GDAL binding.

gdal_rasterize

burns vector geometries into a raster

SYNOPSIS

```
Usage: gdal_rasterize [-b band]* [-i] [-at]
       [-burn value]* | [-a attribute_name] [-3d]
       [-l layername]* [-where expression] [-sql select_statement]
       [-of format] [-a_srs srs_def] [-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}] [-q]
       <src_datasource> <dst_filename>
```

DESCRIPTION

This program burns vector geometries (points, lines and polygons) into the raster band(s) of a raster image. Vectors are read from OGR supported vector formats.

Note that the vector data must in the same coordinate system as the raster data; on the fly reprojection is not provided.

Since GDAL 1.8.0, the target GDAL file can be created by gdal_rasterize. One of -tr or -ts option must be used in that case.

-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.

-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 a polygon.

-at:
: Enables the ALL_TOUCHED rasterization option so that all pixels touched by lines or polygons will be updated not just those one 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. These values are adjusted by the burn value given by "-burn value" or "-a attribute_name" if provided. As of now, only points and lines are drawn in 3D.

-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.

-of *format*:
: (GDAL >= 1.8.0) Select the output format. The default is GeoTIFF (GTiff). Use the short format name.

-a_nodata *value*:
: (GDAL >= 1.8.0) Assign a specified nodata value to output bands.

-init *value*:
: (GDAL >= 1.8.0) 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*:
: (GDAL >= 1.8.0) Override the projection for the output file. If not specified, the projection of the input vector file will be used if available. If incompatible projections between input and output files, no attempt will be made to reproject features. The *srs_def* may be any of the usual GDAL/OGR forms, complete WKT, PROJ.4, EPSG:n or a file containing the WKT.

-co *"NAME=VALUE"*:
: (GDAL >= 1.8.0) Passes a creation option to the output format driver. Multiple **-co** options may be listed. See format specific documentation for legal creation options for each format.

-te *xmin ymin xmax ymax* :
: (GDAL >= 1.8.0) 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* :
: (GDAL >= 1.8.0) set target resolution. The values must be expressed in georeferenced units. Both must be positive values.

-tap:
: (GDAL >= 1.8.0) (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*:
: (GDAL >= 1.8.0) set output file size in pixels and lines. Note that -ts cannot be used with -tr

-ot *type*:
: (GDAL >= 1.8.0) For the output bands to be of the indicated data type. Defaults to Float64

-q:
: (GDAL >= 1.8.0) 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. Before GDAL 1.8.0, gdal_rasterize could not create new output files.

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
```

gdaltransform

transforms coordinates

SYNOPSIS

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

DESCRIPTION

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

-s_srs *srs def*:
: source spatial reference set. The coordinate systems that can be passed are anything supported by the OGRSpatialReference.SetFromUserInput() call, which includes EPSG PCS and GCSes (ie. EPSG:4296), PROJ.4 declarations (as above), or the name of a .prf file containing well known text.

-t_srs *srs_def*:
: target spatial reference set. The coordinate systems that can be passed are anything supported by the OGRSpatialReference.SetFromUserInput() call, which includes EPSG PCS and GCSes (ie. EPSG:4296), PROJ.4 declarations (as above), or the name of a .prf file containing well known text.

-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)

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 (or triples) 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.

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
```

nearblack

convert nearly black/white borders to black

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
```

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 airphotos so that color pixels can be treated as transparent when mosaicing.

-o *outfile*:
: The name of the output file to be created. Newly created files are created with the HFA driver by default (Erdas Imagine - .img)

-of *format*:
: (GDAL 1.8.0 or later) Select the output format. Use the short format name (GTiff for GeoTIFF for examle).

-co *"NAME=VALUE"*:
: (GDAL 1.8.0 or later) Passes a creation option to the output format driver. Multiple **-co** options may be listed. See 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*::
: (GDAL >= 1.9.0) 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:
: (GDAL 1.8.0 or later) 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:
: (GDAL 1.8.0 or later) 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:
: (GDAL 1.8.0 or later) 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.

gdal_retile.py

gdal_retile.py retiles a set of tiles and/or build tiled pyramid levels

Usage:

```
gdal_retile.py [-v] [-co NAME=VALUE]* [-of out_format] [-ps pixelWidth pixelHeight]
               [-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]
               -targetDir TileDirectory input_files
```

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 subdirs numbered from 1. Created tile names have a numbering schema and contain the name of the source tiles(s)

-of *format*:
: Output format, defaults to GeoTIFF (GTiff).

-co *NAME=VALUE*:
: Creation option for output file. Multiple options can be specified.

-ot *datatype*:
: Force the output image bands to have a specific type. Use type names (ie. 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

-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 (ie.EPSG:4296), PROJ.4 declarations (as above), or the name of a .prf 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 propageted to created tiles (if possible) and to the optional shape file(s)

-tileIndex *tileIndexName*:
: The name of shape file containing the result tile(s) index

-tileIndexField *tileIndexFieldName*:
: The name of the attribute containing the tile name

-csv *csvFileName*:
: The name of the csv file containing the tile(s) georeferencing information. The file contains 5 columns:

tilename,minx,maxx,miny,maxy

-csvDelim *column delimiter*:
: The column delimter 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 to 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 subdirectory called 0, the pyramids in subdirectories numbered 1,2,.... Within each of these directories another level of subdirectories is created, numbered from 0...n, depending of how many tile rows are needed for each level. Finally, a directory contains only the the tiles for one row for a specific level. For large images a performance improvement of a factor N could be achieved.

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

gdal_grid

creates regular grid from the scattered data

SYNOPSIS

```
gdal_grid [-ot {Byte/Int16/UInt16/UInt32/Int32/Float32/Float64/
          CInt16/CInt32/CFloat32/CFloat64}]
          [-of format] [-co "NAME=VALUE"]
          [-zfield field_name]
          [-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]
          [-a algorithm[:parameter1=value1]*] [-q]
          <src_datasource> <dst_filename>
```

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.

-ot *type*:
: For the output bands to be of the indicated data type.

-of *format*:
: Select the output format. The default is GeoTIFF (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 *srs_def* may be any of the usual GDAL/OGR forms, complete WKT, PROJ.4, EPSG:n or a file containing the WKT.

-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).

-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"*:
: Passes a creation option to the output format driver. Multiple **-co** options may be listed. See 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.

INTERPOLATION ALGORITHMS

There are number of interpolation algorithms to choose from.

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 radiuses 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 radiuses are non-zero). 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).

DATA METRICS

Besides the interpolation functionality gdal_grid can be used to compute some data metrics using the specified window and output grid geometry. These metrics are:

minimum:
: Minimum value found in grid node search ellipse.

maximum:
: Maximum value found in grid node search ellipse.

range:
: A difference between the minimum and maximum values found in grid node search ellipse.

count:
: A number of data points found in grid node search ellipse.

average_distance:
: An average distance between the grid node (center of the search ellipse) and all of the data points found in grid node search ellipse.

average_distance_pts:

An average distance between the data points found in grid node search ellipse. The distance between each pair of points within ellipse is calculated and average of all distances is set as a grid node value.

All the metrics have the same set of options:

radius1:
: The first radius (X axis if rotation angle is 0) of search ellipse. Set this parameter to zero to use whole point array. Default is 0.0.

radius2:
: The second radius (Y axis if rotation angle is 0) of search ellipse. Set this parameter to zero to use whole point array. Default is 0.0.

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 radiuses are non-zero). Default is 0.

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

READING COMMA SEPARATED VALUES

Often you have a text file with a list of comma separated XYZ values to work with (so called CSV file). You can easily use that kind of data source in gdal_grid. All you need is create a virtual dataset header (VRT) for you CSV file and use it as input datasource for gdal_grid. You can find details on VRT format at Virtual Format description page.

Here is a small example. Let we have a CSV file called *dem.csv* containing

```
Easting,Northing,Elevation
86943.4,891957,139.13
87124.3,892075,135.01
86962.4,892321,182.04
87077.6,891995,135.01
...
```

For above data we will create *dem.vrt* header with the following content:

```
<OGRVRTDataSource>
    <OGRVRTLayer name="dem">
        <SrcDataSource>dem.csv</SrcDataSource>
        <GeometryType>wkbPoint</GeometryType>
        <GeometryField encoding="PointFromColumns" x="Easting" y="Northing" z="Elevation"/>
    </OGRVRTLayer>
</OGRVRTDataSource>
```

This description specifies so called 2.5D geometry with three coordinates X, Y and Z. Z value will be used for interpolation. Now you can use *dem.vrt* with all OGR programs (start with ogrinfo to test that everything works fine). The datasource will contain single layer called *"dem"* 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"/>
```

Comma Separated Value description page contains details on CSV format supported by GDAL/OGR.

EXAMPLE

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 previos one, but reads values to interpolate from the attribute field specified with **-zfield** option instead of geometry record. So in this case X and Y coordinates are being taken from geometry and Z is being taken from the *"Elevation"* field.

```
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
```

gdal_proximity.py

produces a raster proximity map

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] [-fixed-buf-val n]
```

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. The default is GeoTIFF (GTiff). Use the short format name.

-co *"NAME=VALUE"*:
: passes a creation option to the output format driver. Multiple **-co** options may be listed. See format specific documentation for legal creation options for each format.

-ot *datatype*:
: Force the output image bands to have a specific type. Use type names (ie. 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. All pixels beyond this distance will be assigned either the nodata value, or 65535. Distance is interpreted in pixels unless -distunits GEO is specified.

-nodata *n*:
: Specify a nodata value to use for the destination proximity raster.

-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.

---

gdal_polygonize.py

produces a polygon feature layer from a raster

SYNOPSIS

```
gdal_polygonize.py [-o name=value] [-nomask] [-mask filename] raster_file [-b band]
                [-q] [-f ogr_format] out_file [layer] [fieldname]
```

DESCRIPTION

This utility creates vector polygons for all connected regions of pixels in the raster sharing a common pixel value. Each polygon is created with an attribute indicating the pixel value of that polygon. A raster mask may also be provided to determine which pixels are eligible for processing.

The utility will create the output vector datasource if it does not already exist, defaulting to GML format.

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

-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).

raster_file
: The source raster file from which polygons are derived.

-b *band*:
: The band on *raster_file* to build the polygons from.

-f *ogr_format*
: Select the output format of the file to be created. Default is GML.

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").

-o *name=value*:
: Specify a special argument to the algorithm. Currently none are supported.

-q:
: The script runs in quiet mode. The progress monitor is supressed and routine messages are not displayed.

gdal_sieve.py

removes small raster polygons

SYNOPSIS

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

DESCRIPTION

The gdal_sieve.py script removes raster polygons smaller than a provided threshold size (in pixels) and replaces 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.

Additional details on the algorithm are available in the GDALSieveFilter() docs.

-q:
: The script runs in quiet mode. The progress monitor is supressed and routine messages are not displayed.

-st *threshold*:
: Set the size threshold in pixels. Only raster polygons smaller than this size will be removed.

-o *name=value*:
: Specify a special argument to the algorithm. Currently none are supported.

-4:
: Four connectedness should be used when determining polygons. That is diagonal pixels are not considered directly connected. This is the default.

-8:
: Eight connectedness should be used when determining polygons. That is diagonal pixels are considered directly connected.

srcfile
: The source raster file used to identify target pixels. Only the first 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 filtered result. If not provided, the source band is updated in place.

-of *format*:
: Select the output format. The default is GeoTIFF (GTiff). Use the short format name.

---

gdal_fillnodata.py

fill raster regions by interpolation from edges

SYNOPSIS

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

DESCRIPTION

The 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 supressed and routine messages are not displayed.

-md *max_distance*:
: The maximum distance (in pixels) that the algorithm will search out for values to interpolate.

-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 GeoTIFF (GTiff). Use the short format name.

---

gdallocationinfo

raster query tool

SYNOPSIS

Usage:

```
Usage: gdallocationinfo [--help-general] [-xml] [-lifonly] [-valonly]
                        [-b band]* [-l_srs srs_def] [-geoloc] [-wgs84]
                        srcfile [x y]
```

DESCRIPTION

The gdallocationinfo utility provide a mechanism to query information about a pixel given it's 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 (ie. 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.

-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.

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 three things:

- The location of the pixel in pixel/line space.
- The result of a LocationInfo metadata query against the datasource - currently this is only implemented for VRT files which will report the file(s) used to satisfy requests for that pixel.
- 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 commandline, or read from stdin. More than one coordinate pair can be supplied when reading coordinatesis 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.

EXAMPLE

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>
```

gdal-config

determines various information about a GDAL installation

SYNOPSIS

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

DESCRIPTION

This utility script (available on Unix systems) can be used to determine various information about a GDAL installation. It is normally just used by configure scripts for applications using GDAL but can be queried by an end user.

--prefix:
: the top level directory for the GDAL installation.

--libs:
: The libraries and link directives required to use GDAL.

--cflags:
: The include and macro definition required to compiled modules using GDAL.

--version:
: Reports the GDAL version.

--ogr-enabled:
: Reports "yes" or "no" to standard output depending on whether OGR is built into GDAL.

--formats:
: Reports which formats are configured into GDAL to stdout.

GDAL Raster Formats

Long Format Name	Code	Creation	Georeferencing	Maximum file size[1]	Compiled by default
Arc/Info ASCII Grid	AAIGrid	Yes	Yes	2GB	Yes
ACE2	ACE2	No	Yes	--	Yes
ADRG/ARC Digitilized Raster Graphics (.gen/.thf)	ADRG	Yes	Yes	--	Yes
Arc/Info Binary Grid (.adf)	AIG	No	Yes	--	Yes
AIRSAR Polarimetric	AIRSAR	No	No	--	Yes
Magellan BLX Topo (.blx, .xlb)	BLX	Yes	Yes	--	Yes
Bathymetry Attributed Grid (.bag)	BAG	No	Yes	2GiB	No, needs libhdf5
Microsoft Windows Device Independent Bitmap (.bmp)	BMP	Yes	Yes	4GiB	Yes
BSB Nautical Chart Format (.kap)	BSB	No	Yes	--	Yes, can be disabled
VTP Binary Terrain Format (.bt)	BT	Yes	Yes	--	Yes
CEOS (Spot for instance)	CEOS	No	No	--	Yes
DRDC COASP SAR Processor Raster	COASP	No	No	--	Yes
TerraSAR-X Complex SAR Data Product	COSAR	No	No	--	Yes
Convair PolGASP data	CPG	No	Yes	--	Yes
USGS LULC Composite Theme Grid	CTG	No	Yes	--	Yes
Spot DIMAP (metadata.dim)	DIMAP	No	Yes	--	Yes
ELAS DIPEx	DIPEx	No	Yes	--	Yes
DODS / OPeNDAP	DODS	No	Yes	--	No, needs libdap
First Generation USGS DOQ (.doq)	DOQ1	No	Yes	--	Yes
New Labelled USGS DOQ (.doq)	DOQ2	No	Yes	--	Yes
Military Elevation Data (.dt0, .dt1, .dt2)	DTED	Yes	Yes	--	Yes
Arc/Info Export E00 GRID	E00GRID	No	Yes	--	Yes
ECRG Table Of Contents (TOC.xml)	ECRGTOC	No	Yes	--	Yes
ERDAS Compressed Wavelets (.ecw)	ECW	Yes	Yes		No, needs ECW SDK
ESRI .hdr Labelled	EHdr	Yes	Yes	No limits	Yes
Erdas Imagine Raw	EIR	No	Yes	--	Yes
NASA ELAS	ELAS	Yes	Yes	--	Yes
ENVI .hdr Labelled Raster	ENVI	Yes	Yes	No limits	Yes

Epsilon - Wavelet compressed images	EPSILON	Yes	No	--	No, needs EPSILON library
ERMapper (.ers)	ERS	Yes	Yes		Yes
Envisat Image Product (.n1)	ESAT	No	No	--	Yes
EOSAT FAST Format	FAST	No	Yes	--	Yes
FIT	FIT	Yes	No	--	Yes
FITS (.fits)	FITS	Yes	No	--	No, needs libcfitsio
Fuji BAS Scanner Image	FujiBAS	No	No	--	Yes
Generic Binary (.hdr Labelled)	GENBIN	No	No	--	Yes
Oracle Spatial GeoRaster	GEORASTER	Yes	Yes	--	No, needs Oracle client libraries
GSat File Format	GFF	No	No	--	Yes
Graphics Interchange Format (.gif)	GIF	Yes	No	2GB	Yes (internal GIF library provided)
WMO GRIB1/GRIB2 (.grb)	GRIB	No	Yes	2GB	Yes, can be disabled
GMT Compatible netCDF	GMT	Yes	Yes	2GB	No, needs libnetcdf
GRASS Rasters	GRASS	No	Yes	--	No, needs libgrass
GRASS ASCII Grid	GRASSASCIIGrid	No	Yes	--	Yes
Golden Software ASCII Grid	GSAG	Yes	No	--	Yes
Golden Software Binary Grid	GSBG	Yes	No	4GiB (32767x32767 of 4 bytes each + 56 byte header)	Yes
Golden Software Surfer 7 Binary Grid	GS7BG	No	No	4GiB	Yes
GSC Geogrid	GSC	Yes	No	--	Yes
TIFF / BigTIFF / GeoTIFF (.tif)	GTiff	Yes	Yes	4GiB for classical TIFF / No limits for BigTIFF	Yes (internal libtiff and libgeotiff provided)
NOAA .gtx vertical datum shift	GTX	Yes	Yes		Yes
GXF - Grid eXchange File	GXF	No	Yes	4GiB	Yes
Hierarchical Data Format Release 4 (HDF4)	HDF4	Yes	Yes	2GiB	No, needs libdf
Hierarchical Data Format Release 5 (HDF5)	HDF5	No	Yes	2GiB	No, needs libhdf5
HF2/HFZ heightfield raster	HF2	Yes	Yes	-	Yes
Erdas Imagine (.img)	HFA	Yes	Yes	No limits[2]	Yes
	IDA	Yes	Yes	2GB	Yes

Image Display and Analysis (WinDisp)					
ILWIS Raster Map (.mpr,.mpl)	ILWIS	Yes	Yes	--	Yes
Intergraph Raster	INGR	Yes	Yes	2GiB	Yes
USGS Astrogeology ISIS cube (Version 2)	ISIS2	Yes	Yes	--	Yes
USGS Astrogeology ISIS cube (Version 3)	ISIS3	No	Yes	--	Yes
JAXA PALSAR Product Reader (Level 1.1/1.5)	JAXAPALSAR	No	No	--	Yes
Japanese DEM (.mem)	JDEM	No	Yes	--	Yes
JPEG JFIF (.jpg)	JPEG	Yes	Yes	4GiB (max dimentions 65500x65500)	Yes (internal libjpeg provided)
JPEG-LS	JPEGLS	Yes	No	--	No, needs CharLS library
JPEG2000 (.jp2, .j2k)	JPEG2000	Yes	Yes	2GiB	No, needs libjasper
JPEG2000 (.jp2, .j2k)	JP2ECW	Yes	Yes	500MB	No, needs ECW SDK
JPEG2000 (.jp2, .j2k)	JP2KAK	Yes	Yes	No limits	No, needs Kakadu library
JPEG2000 (.jp2, .j2k)	JP2MrSID	Yes	Yes		No, needs MrSID SDK
JPEG2000 (.jp2, .j2k)	JP2OpenJPEG	Yes	Yes		No, needs OpenJPEG library (v2)
JPIP (based on Kakadu)	JPIPKAK	No	Yes		No, needs Kakadu library
KMLSUPEROVERLAY	KMLSUPEROVERLAY	Yes	Yes		Yes
NOAA Polar Orbiter Level 1b Data Set (AVHRR)	L1B	No	Yes	--	Yes
Erdas 7.x .LAN and .GIS	LAN	No	Yes	2GB	Yes
FARSITE v.4 LCP Format	LCP	No	Yes	No limits	Yes
Daylon Leveller Heightfield	Leveller	No	Yes	2GB	Yes
NADCON .los/.las Datum Grid Shift	LOSLAS	No	Yes		Yes
In Memory Raster	MEM	Yes	Yes		Yes
Vexcel MFF	MFF	Yes	Yes	No limits	Yes
Vexcel MFF2	MFF2 (HKV)	Yes	Yes	No limits	Yes
MG4 Encoded Lidar	MG4Lidar	No	Yes	--	No, needs LIDAR SDK
Multi-resolution Seamless Image Database	MrSID	No	Yes	--	No, needs MrSID SDK
Meteosat Second Generation	MSG	No	Yes		

					No, needs msg library
EUMETSAT Archive native (.nat)	MSGN	No	Yes		Yes
NLAPS Data Format	NDF	No	Yes	No limits	Yes
NITF (.ntf, .nsf, .gn?, .hr?, .ja?, .jg?, .jn?, .lf?, .on?, .tl?, .tp?, etc.)	NITF	Yes	Yes	10GB	Yes
NetCDF	netCDF	Yes	Yes	2GB	No, needs libnetcdf
NTv2 Datum Grid Shift	NTv2	Yes	Yes		Yes
Northwood/VerticalMapper Classified Grid Format .grc/.tab	NWT_GRC	No	Yes	--	Yes
Northwood/VerticalMapper Numeric Grid Format .grd/.tab	NWT_GRD	No	Yes	--	Yes
OGDI Bridge	OGDI	No	Yes	--	No, needs OGDI library
OZI OZF2/OZFX3	OZI	No	Yes	--	No
PCI .aux Labelled	PAux	Yes	No	No limits	Yes
PCI Geomatics Database File	PCIDSK	Yes	Yes	No limits	Yes
PCRaster	PCRaster	Yes	Yes		Yes (internal libcsf provided)
Geospatial PDF	PDF	No	Yes	--	No, needs libpoppler or libpodofo
NASA Planetary Data System	PDS	No	Yes	--	Yes
Portable Network Graphics (.png)	PNG	Yes	No		Yes (internal libpng provided)
PostGIS Raster (previously WKTRaster)	PostGISRaster	No	Yes	--	No, needs PostgreSQL library
Netpbm (.ppm,.pgm)	PNM	Yes	No	No limits	Yes
R Object Data Store	R	Yes	No	--	Yes
Rasdaman	RASDAMAN	No	No	No limits	No (needs raslib)
Rasterlite - Rasters in SQLite DB	Rasterlite	Yes	Yes	--	No (needs OGR SQLite driver)
Swedish Grid RIK (.rik)	RIK	No	Yes	4GB	Yes (internal zlib is used if necessary)
Raster Matrix Format (*.rsw, .mtw)	RMF	Yes	Yes	4GB	Yes
	RPFTOC	No	Yes	--	Yes

Raster Product Format/RPF (CADRG, CIB)					
RadarSat2 XML (product.xml)	RS2	No	Yes	4GB	Yes
Idrisi Raster	RST	Yes	Yes	No limits	Yes
SAGA GIS Binary format	SAGA	Yes	Yes	--	Yes
SAR CEOS	SAR_CEOS	No	Yes	--	Yes
ArcSDE Raster	SDE	No	Yes	--	No, needs ESRI SDE
USGS SDTS DEM (*CATD.DDF)	SDTS	No	Yes	--	Yes
SGI Image Format	SGI	Yes	Yes	--	Yes
Snow Data Assimilation System	SNODAS	No	Yes	--	Yes
Standard Raster Product (ASRP/USRP)	SRP	No	Yes	2GB	Yes
SRTM HGT Format	SRTMHGT	Yes	Yes	--	Yes
Terragen Heightfield (.ter)	TERRAGEN	Yes	No	--	Yes
EarthWatch/DigitalGlobe .TIL	TIL	No	No	--	Yes
TerraSAR-X Product	TSX	Yes	No	--	Yes
USGS ASCII DEM / CDED (.dem)	USGSDEM	Yes	Yes	--	Yes
GDAL Virtual (.vrt)	VRT	Yes	Yes	--	Yes
OGC Web Coverage Service	WCS	No	Yes	--	No, needs libcurl
WEBP	WEBP	Yes	No	--	No, needs libwebp
OGC Web Map Service	WMS	No	Yes	--	No, needs libcurl
X11 Pixmap (.xpm)	XPM	Yes	No		Yes
ASCII Gridded XYZ	XYZ	Yes	Yes	--	Yes
ZMap Plus Grid	ZMap	Yes	Yes		Yes

[1]Maximum file size is not only determined by the file format itself, but operating system/file system capabilities as well. Look here for details.

[2]ERDAS Imagine has different file format for large files, where 32-bit pointers cannot be used. Look for details here.

$Id: formats_list.html 22861 2011-08-04 20:13:27Z rouault $

AIRSAR -- AIRSAR Polarimetric Format

Most variants of the AIRSAR Polarimetric Format produced by the AIRSAR Integrated Processor are supported for reading by GDAL. AIRSAR products normally include various associated data files, but only the imagery data themselves is supported. Normally these are named *mission*_l.dat (L-Band) or *mission*_c.dat (C-Band).

AIRSAR format contains a polarimetric image in compressed stokes matrix form. Internally GDAL decompresses the data into a stokes matrix, and then converts that form into a covariance matrix. The returned six bands are the six values needed to define the 3x3 Hermitian covariance matrix. The convention used to represent the covariance matrix in terms of the scattering matrix elements HH, HV (=VH), and VV is indicated below. Note that the non-diagonal elements of the matrix are complex values, while the diagonal values are real (though represented as complex bands).

- Band 1: Covariance_11 (Float32) = HH*conj(HH)
- Band 2: Covariance_12 (CFloat32) = sqrt(2)*HH*conj(HV)
- Band 3: Covariance_13 (CFloat32) = HH*conj(VV)
- Band 4: Covariance_22 (Float32) = 2*HV*conj(HV)
- Band 5: Covariance_23 (CFloat32) = sqrt(2)*HV*conj(VV)
- Band 6: Covariance_33 (Float32) = VV*conj(VV)

The identities of the bands are also reflected in metadata and in the band descriptions.

The AIRSAR product format includes (potentially) several headers of information. This information is captured and represented as metadata on the file as a whole. Information items from the main header are prefixed with "MH_", items from the parameter header are prefixed with "PH_" and information from the calibration header are prefixed with "CH_". The metadata item names are derived automatically from the names of the fields within the header itself.

No effort is made to read files associated with the AIRSAR product such as *mission*_l.mocomp, *mission*_meta.airsar or *mission*_meta.podaac.

See Also:

- AIRSAR Data Format

BAG --- Bathymetry Attributed Grid

This driver provides read-only support for bathymetry data in the BAG format. BAG files are actually a specific product profile in an HDF5 file, but a custom driver exists to present the data in a more convenient manner than is available through the generic HDF5 driver.

BAG files have two 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.

The full XML metadata is available in the "xml:BAG" metadata domain.

Nodata, minimum and maximum values for each band are also provided.

See Also:

- Implemented as `gdal/frmts/hdf5/bagdataset.cpp`.
- The Open Navigation Surface Project

BLX -- Magellan BLX Topo File Format

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.

Georeferencing

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

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

BMP --- Microsoft Windows Device Independent Bitmap

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.

Creation Options

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

See Also:

- Implemented as `gdal/frmts/bmp/bmpdataset.cpp`.
- MSDN Bitmap Reference

COSAR -- TerraSAR-X Complex SAR Data Product

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 bytes real, 16 bytes 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.

See Also:

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

DODS -- OPeNDAP Grid Client

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

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 percipitate[scan = 5][longitude = 360][latitude = 80];
      Float64 relError[scan = 5][longitude = 360][latitude = 80];
    } PlanetaryGrid;
  } DATA_GRANULE;
} 3B42.HDF;
```

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

http://maps.gdal.org/daac-bin/nph-hdf/3B42.HDF?DATA_GRANULE.PlanetaryGrid.percipitate[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 seperate bands.

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 AIX 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[\"WGS84\",DATUM[\"WGS_1984\",SPHEROID[\"WGS84\",6378137,298.257223563]],
           PRIMEM[\"Greenwich\",0],UNIT[\"degree\",0.0174532925199433]]";0
        Metadata {
          String TIFFTAG_XRESOLUTION "400";
          String TIFFTAG_YRESOLUTION "400";
          String TIFFTAG_RESOLUTIONUNIT "2 (pixels/inch)";
        }
    }
    band_1 {
        String Description "...";
        String
    }
}
```

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 seperated by sapces. 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 interprete by a user than the GeoTransform. They will also be used to compute a GeoTransform internally if one is note provided, but if both are provided the GeoTransform will take precidence.

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;
    Byte green 0;
    Byte blue 0;
    Byte alpha 255;
```

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

See Also:

- OPeNDAP Website

DTED -- Military Elevation Data

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.

Read Issues

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.

Georeferencing Issues

The DTED specification (MIL-PRF-89020B) states that *horizontal datum shall be the World Geodetic System (WGS 84)*. 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.

Checksum Issues

The default behaviour 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.

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.

See Also:

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

ECW -- ERDAS Compress Wavelets (.ecw)

GDAL supports .ecw format for read access and write. The current implementation reads any number of bands but returns only as eight bit image data. Coordinate system and georeferencing transformations are read, but in some cases coordinate systems may not translate.

Support for the ECW driver in GDAL is optional, and requires linking in external ECW SDK libraries provided by ERDAS.

In addition to ECW files, this driver also supports access to network image services using the "ECWP" protocol. Use the full ecwp:// url of the service as the dataset name. When built with SDK 4.1 or newer it is also possible to take advantage of RFC 24 style asynchronous access to ECWP services.

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.

Creation Issues

The ECW 4.x SDK from ERDAS 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.

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.

Creation Options:

- **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.
- **LARGE_OK=YES**: When built with the ECW 3.x SDK this option can be set to allow compressing files larger than 500MB. It is the users responsibility to ensure that the licensing requirments for large file compression are being adhered to.
- **ECW_ENCODE_KEY=key**: Provide the OEM encoding key purchased from Erdas which permits encoding images. The key is is approximately 129 hex digits long. It may also be provided globally as a configuration option.
- **ECW_ENCODE_COMPANY=name**: Provide the name of the company ERDAS issued the OEM encoding key (see ECW_ENCODE_KEY) to. This must exactly match the name used by ERDAS in issuing the OEM key. It may also be provided globally as a configuration option.

ECW format does not support creation of overviews since the ECW format is already considered to be optimized for "arbitrary overviews".

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.
- **ECW_ENCODE_KEY, ECW_ENCODE_COMPANY**: These values, as described in the creation options, may also be set as configuration options. See above.

See Also:

- Implemented in `gdal/frmts/ecw`.
- ECW SDK available at erdas.com.
- GDAL ECW Build Hints

ELAS - Earth Resources Laboratory Applications Software

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.

See Also:

- Short announcement of ELAS driver in GDAL
- NASA Software Used In Imaging Service includes ELAS overview

Epsilon - Wavelet compressed images

Starting with GDAL 1.7.0, GDAL can read and write wavelet-compressed images through the Epsilon library. Starting with GDAL 1.9.0, 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 intented to be used by the Rasterlite driver.

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 artefacts 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

See Also:

- EPSILON home page

ERS -- ERMapper .ERS

GDAL supports reading and writing raster files with .ers header files, with some limitations. The .ers ascii format is used by ERMapper for labelling 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.

See Also:

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

FAST -- EOSAT FAST Format

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.

Data

FAST-L7A

FAST-L7A consists form several files: big ones with image data and three small files with administrative information. You should give to driver one of the administrative files:

- L7fppprrr_rrrYYYYMMDD_HPN.FST: panchromatic band header file with 1 band
- L7fppprrr_rrrYYYYMMDD_HRF.FST: VNIR/ SWIR bands header file with 6 bands
- L7fppprrr_rrrYYYYMMDD_HTM.FST: thermal bands header file with 2 bands

All raw images corresponded to their administrative files will be imported as GDAL bands.

From the `` Level 1 Product Output Files Data Format Control Book'':

```
The file naming convention for the FAST-L7A product files is
L7fppprrr_rrrYYYYMMDD_AAA.FST

where

L7 = Landsat 7 mission

f = ETM+ format (1 or 2) (data not pertaining to a specific format defaults
to 1)

ppp = starting path of the product

rrr_rrr = starting and ending rows of the product

YYYYMMDD = acquisition date of the image

AAA = file type:
HPN = panchromatic band header file
HRF = VNIR/ SWIR bands header file
HTM = thermal bands header file
B10 = band 1
B20 = band 2
B30 = band 3
B40 = band 4
B50 = band 5
```

```
B61 = band 6L
B62 = band 6H
B70 = band 7
B80 = band 8

FST = FAST file extension
```

So you should give to driver one of the `L7fppprrr_rrrYYYYMMDD_HPN.FST`, `L7fppprrr_rrrYYYYMMDD_HRF.FST` or `L7fppprrr_rrrYYYYMMDD_HTM.FST` files.

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.

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.

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**.

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 ESDIS Level 1 Product Generation System (LPGS) Output Files DFCB, Vol. 5, Book 2)
- EOSAT Fast Format REV. C description available from http://www.euromap.de/docs/doc_001.html

Oracle Spatial GeoRaster

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, it's 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 note 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 bellow)

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
```

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, JPEG-B, DEFLATE or NONE. The two JPEG options are lossy, meaning that the original pixel values are changed. The JPEG-F store a full JPEG structure on each block while JPEG-B are smaller since they don't store Huffman and Quantization tables.
- QUALITY: Quality compression option for JPEG ranging from 0 to 100. The default is 75.
- NBITS: Sub byte data type, options: 1, 2 or 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-B"
```

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"
```

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

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=101 out.tif
% gdal_translate geor:spock/lion@orcl,gdal_rdt,101 out.tif
```

Note: On the above example that acessing by RDT/RID doesn't need schame name as long as the users is granted full acess to both tables.

Loading:

```
% gdal_translate -of georaster input.tif geor:spock/lion@orcl,scott.
% gdal_translate -of georaster input.tif geor:spock/lion@orcl,scott.cities,image \
  -co INSERT="(1,'Rio de Janeiro',sdo_geor.init('cities_rdt'))"
```

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
```

GIF -- Graphics Interchange Format

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.

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.

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. Tranparent 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.

See Also:

- giflib Home Page

GRASS -- GRASS Rasters

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

GRASS rasters can be selected in several ways.

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

   eg.

   ```
   % 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.

   eg.

   ```
   % gdalinfo /usr2/data/grassdb/imagery/raw/group/testmff/REF
   % gdalinfo /usr2/data/grassdb/imagery/raw/group/testmff
   ```
3. If there is a correct .grassrc5 setup in the users home directory then cells or imagery groups may be opened just by the cell name. This only works for cells or images in the current location and mapset as defined in the .grassrc5 file.

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

- Up to 256 entries from cell colormaps are read (0-255).
- Compressed and uncompressed integer, floating point and double precision cells are all supported. Integer cells 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 cells are treated as UInt32.
- Georeferencing information is properly read from GRASS.
- An attempt is made to translate coordinate systems, but some conversions may be flawed, in particular in handling of datums and units.

See Also:

- GRASS GIS Home Page
- libgrass page

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.

GRIB -- WMO General Regularly-distributed Information in Binary form

GDAL supports reading of GRIB1 and GRIB2 format raster data, with some degree of support for 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. The JasPer library generally provides the best jpeg2000 support for the GRIB driver.

GRIB files may a be represented in GDAL as having many bands, with some sets of bands representing a time sequence. GRIB bands are represented as Float64 (double precision floating point) regardless of the actual values. GRIB metadata is captured as per-band metadata and used to set band descriptions, similar to this:

```
  Description = 100000[Pa] ISBL="Isobaric surface"
    GRIB_UNIT=[gpm]
    GRIB_COMMENT=Geopotential height [gpm]
    GRIB_ELEMENT=HGT
    GRIB_SHORT_NAME=100000-ISBL
    GRIB_REF_TIME=  1201100400 sec UTC
    GRIB_VALID_TIME=  1201104000 sec UTC
    GRIB_FORECAST_SECONDS=3600 sec
```

GRIB2 files may also include an extract of the product definition template number (octet 8-9), and the product definition template values (octet 10+) as metadata like this:

```
    GRIB_PDS_PDTN=0
    GRIB_PDS_TEMPLATE_NUMBERS=3 5 2 0 105 0 0 0 1 0 0 0 1 100 0 0 1 134 160 255 0 0 0 0 0
```

Known issues:

The library that GDAL uses to read GRIB files is known to be not thread-safe, so you should avoid reading or writing several GRIB datasets at the same time from different threads.

See Also:

- NOAA NWS NDFD "degrib" GRIB2 Decoder
- NOAA NWS NCEP g2clib grib decoding library
- WMS GRIB Format Documents

GTiff -- GeoTIFF File Format

Most forms of TIFF and GeoTIFF files are supported by GDAL for reading, and somewhat less varieties can be written.

When built with internal libtiff or with libtiff >= 4.0, 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.

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 EPSG .csv files available. They must be found at the location pointed to by the GEOTIFF_CSV 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 extention .tfw, .tifw/.tiffw or .wld, as well as a MapInfo .tab file (only control points used, Coordsys ignored).

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 RPC Georeferencing RFC. If .RPB or _RPC.TXT files are found, they will be used to read the RPCs, even if the *RPCCoefficientTag* tag is set.

Internal nodata masks

(from GDAL 1.6.0)

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

Starting with GDAL 1.8.0, 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).

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.

(From GDAL 1.6.0) If a GeoTIFF file has a transparency mask and the GDAL_TIFF_INTERNAL_MASK environment variable is set to YES and the GeoTIFF file is opened in update mode, BuildOverviews() will automatically create overviews for the internal transparency mask. These overviews will be refreshed by further calls to BuildOverviews() even if GDAL_TIFF_INTERNAL_MASK is not set to YES.

(From GDAL 1.8.0) 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 value is 128.

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)

The name of the metadata item to use is one of the above names ("TIFFTAG_DOCUMENTNAME", ...).

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 stored as XML raw content in the xml:XMP metadata domain.

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.

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 with exactly 3 bands will be given a photometric interpretation of RGB, files with exactly four bands will have a photometric interpretation of RGBA, while all other combinations will have a photometric interpretation of MIN_IS_WHITE. Files with pseudo-color tables, or GCPs can currently only be created when creating from an existing GDAL dataset with those objects (GDALDriver::CreateCopy()).

Note that the GeoTIFF format does not support parametric description of datums, so TOWGS84 parameters in coordinate systems are lost in GeoTIFF format.

Creation Options

- **TFW=YES**: Force the generation of an associated ESRI world file (.tfw).See a World Files section for details.
- **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 stripped 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.
- **COMPRESS=[JPEG/LZW/PACKBITS/DEFLATE/CCITTRLE/CCITTFAX3/CCITTFAX4/NONE]**: 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. None is the default.
- **PREDICTOR=[1/2/3]**: Set the predictor for LZW or DEFLATE compression. The default is 1 (no predictor), 2 is horizontal differencing and 3 is floating point prediction.
- **SPARSE_OK=TRUE/FALSE** (From GDAL 1.6.0): Should newly created files 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. 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.
- **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 6.
- **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**: 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).
- **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 occationally useful when writing files to be read by applications intolerant of unrecognised 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 (uncompressed, and image larger than 4GB).
  - ♦ IF_SAFER will create BigTIFF if the resulting file *might* exceed 4GB.

  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 (which is the case of the internal libtiff version from GDAL >= 1.5.0). The default is IF_NEEDED. (IF_NEEDED and IF_SAFER are available from GDAL 1.6.0).

  When creating a new GeoTIFF with no compression, GDAL computes in advance the size of the resulting file. If that computed file size is over 4GiB, GDAL will automatically decide to create a BigTIFF file. However, when compression is used, it is not possible in advance to known the final size of the file, so classical TIFF will be chosen. In that case, the user must explicitely require the creation of a BigTIFF with BIGTIFF=YES if he anticipates the final file to be too big for classical TIFF format. If BigTIFF creation is not explicitely asked or guessed and that 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]**: (GDAL >= 1.8.0, 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. 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.

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

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 : (GDAL >= 1.9.0) 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 PixelIsPoint is interprated w.r.t geotransform. See RFC 33: GTiff - Fixing PixelIsPoint

Interpretation for more details. Default value : FALSE
- GTIFF_REPORT_COMPD_CS: (GDAL >= 1.9.0). Can be set to TRUE to avoid stripping the vertical CS of compound CS. Default value : FALSE
- 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).

See Also:

- GeoTIFF Information Page
- libtiff Page
- Details on BigTIFF file format

HDF4 --- Hierarchical Data Format Release 4 (HDF4)

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 form `Terra' and `Aqua' satellites). In the future they will switch to HDF5-EOS format, which will be used for telemetry from `Aura' satellite.

Multiple Image Handling (Subdatasets)

Hierarchical Data Format is a container for several different datasets. For data storing Scientific Datasets (SDS) used most often. SDS is a multidimensional array filled by data. One HDF file may contain several different SDS arrays. They may differ in size, number of dimensions and may represent data for different regions.

If the file contains only one SDS that appears to be an image, it may be accessed normally, but if it contains multiple images it may be necessary to import the file via a two step process. The first step is to get a report of the components images (SDS arrays) in the file using **gdalinfo**, and then to import the desired images using gdal_translate. The **gdalinfo** utility lists all multidimensional subdatasets from the input HDF file. The name of individual images (subdatasets) are assigned to the **SUBDATASET_n_NAME** metadata item. The description for each image is found in the **SUBDATASET_n_DESC** metadata item. For HDF4 images the subdataset names will be formatted like this:

HDF4_SDS:subdataset_type:file_name:subdataset_index

where *subdataset_type* shows predefined names for some of the well known HDF datasets, *file_name* is the name of the input file, and *subdataset_index* is the index of the image to use (for internal use in GDAL).

On the second step you should provide this name for **gdalinfo** or **gdal_translate** for actual reading of the data.

For example, we want to read data from the MODIS Level 1B dataset:

```
$ gdalinfo GSUB1.A2001124.0855.003.200219309451.hdf
Driver: HDF4/Hierarchical Data Format Release 4
Size is 512, 512
Coordinate System is `'
Metadata:
  HDFEOSVersion=HDFEOS_V2.7
  Number of Scans=204
  Number of Day mode scans=204
  Number of Night mode scans=0
  Incomplete Scans=0
```

...a lot of metadata output skipped...

```
Subdatasets:
  SUBDATASET_1_NAME=HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:0
  SUBDATASET_1_DESC=[408x271] Latitude (32-bit floating-point)
  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 inte
  SUBDATASET_14_NAME=HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:14
  SUBDATASET_14_DESC=[204x16x10] DC Restore Change for Thermal Bands (8-bit integer)
  SUBDATASET_15_NAME=HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:15
  SUBDATASET_15_DESC=[204x2x40] DC Restore Change for Reflective 250m Bands (8-bit integer)
  SUBDATASET_16_NAME=HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:16
  SUBDATASET_16_DESC=[204x5x20] DC Restore Change for Reflective 500m Bands (8-bit integer)
  SUBDATASET_17_NAME=HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:17
  SUBDATASET_17_DESC=[204x15x10] DC Restore Change for Reflective 1km Bands (8-bit integer)
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0,  512.0)
Upper Right (  512.0,    0.0)
Lower Right (  512.0,  512.0)
Center      (  256.0,  256.0)
```

Now select one of the subdatasets, described as `[12x2040x1354] EV_1KM_RefSB (16-bit unsigned integer)`:

```
$ gdalinfo HDF4_SDS:MODIS_L1B:GSUB1.A2001124.0855.003.200219309451.hdf:2
Driver: HDF4Image/HDF4 Internal Dataset
Size is 1354, 2040
Coordinate System is `'
Metadata:
  long_name=Earth View 1KM Reflective Solar Bands Scaled Integers
```

...metadata skipped...

```
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0, 2040.0)
Upper Right ( 1354.0,    0.0)
Lower Right ( 1354.0, 2040.0)
Center      (  677.0, 1020.0)
Band 1 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
Band 2 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
Band 3 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
Band 4 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
Band 5 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
Band 6 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
Band 7 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
Band 8 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
Band 9 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
Band 10 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
Band 11 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
Band 12 Block=1354x2040 Type=UInt16, ColorInterp=Undefined
```

Or you may use **gdal_translate** for reading image bands from this dataset.

Note that you should provide exactly the contents of the line marked **SUBDATASET_n_NAME** to GDAL, including the **HDF4_SDS:** prefix.

This driver is intended only for importing remote sensing and geospatial datasets in form of raster images. If you want explore all data contained in HDF file you should use another tools (you can find information about different HDF tools using links at end of this page).

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.

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.

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.

Driver building

This driver builded 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).

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

HDF5 --- Hierarchical Data Format Release 5 (HDF5)

This driver intended for HDF5 file formats importing. This modification is suited for use with remote sensing data and fully compatible with underlying HDF5. This driver can import HDF5-EOS files. Currently EOS use HDF5 for data storing (telemetry form `Aura' satellites). In the future they will switch to HDF5 format, which will be used for telemetry from `Aura' satellite.

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
  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/GroundPixel
 (16-bit unsigned integer)
  SUBDATASET_24_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.he5":
//HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/Latitude
  SUBDATASET_24_DESC=[1496x60]  //HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/Latitude
 (32-bit floating-point)
  SUBDATASET_25_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.he5":
//HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/Longitude
  SUBDATASET_25_DESC=[1496x60]  //HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/Longitude
 (32-bit floating-point)
  SUBDATASET_26_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.he5":
//HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/RelativeAzimuthAngle
  SUBDATASET_26_DESC=[1496x60]  //HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/RelativeAzi
 (32-bit floating-point)
  SUBDATASET_27_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.he5":
//HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/SolarAzimuthAngle
  SUBDATASET_27_DESC=[1496x60]  //HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/SolarAzimut
 (32-bit floating-point)
  SUBDATASET_28_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.he5":
//HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/SolarZenithAngle
  SUBDATASET_28_DESC=[1496x60]  //HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/SolarZenith
 (32-bit floating-point)
  SUBDATASET_29_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.he5":
//HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/TerrainHeight
  SUBDATASET_29_DESC=[1496x60]  //HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/TerrainHeig
 (16-bit integer)
  SUBDATASET_30_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.he5":
//HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/ViewingAzimuthAngle
  SUBDATASET_30_DESC=[1496x60]  //HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/ViewingAzim
 (32-bit floating-point)
  SUBDATASET_31_NAME=HDF5:"OMI-Aura_L2-OMTO3_2005m0113t0224-o02648_v002-2005m0625t035355.he5":
//HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/ViewingZenithAngle
  SUBDATASET_31_DESC=[1496x60]  //HDFEOS/SWATHS/OMI_Column_Amount_O3/Geolocation_Fields/ViewingZeni
 (32-bit floating-point)
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0,  512.0)
Upper Right (  512.0,    0.0)
Lower Right (  512.0,  512.0)
Center      (  256.0,  256.0)
```

Now select one of the subdatasets, described as `[1645x60] CloudFraction (32-bit floating-point)`:

```
$ gdalinfo HDF5:"OMI-Aura_L2-OMTO3_2005m0326t2307-o03709_v002-2005m0428t201311.he5":CloudFraction
Driver: HDF5Image/HDF5 Dataset
Size is 60, 1645
Coordinate System is:
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        TOWGS84[0,0,0,0,0,0,0],
        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).

Georeference

There is no universal way of storing georeferencing in HDF files. However, some product types have mechanisms for saving georeferencing, and some of these are supported by GDAL. Currently supported are (*subdataset_type* shown in parenthesis):

- HDF5 OMI/Aura Ozone (O3) Total Column 1-Orbit L2 Swath 13x24km (**Level-2 OMTO3**)

Metadata

No Metadata are read at this time from the HDF5 files.

Driver building

This driver builded on top of NCSA HDF5 library, so you need to dowload prebuild HDF5 librariesI 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 (especialy 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).

See Also:

- Implemented as `gdal/frmts/hdf5/hdf5dataset.cpp` and `gdal/frmts/hdf5/hdf5imagedataset.cpp`.
- The NCSA HDF5 Dowload 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:

- Aura OMI Total Ozone Data Product-OMTO3

HF2 -- HF2/HFZ heightfield raster

(Available for GDAL >= 1.8.0)

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 optionnaly 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 georefencing information through extended header blocks.

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.

See also

- Specification of HF2/HFZ format
- Specification of HF2 extended header blocks

HFA -- Erdas Imagine .img

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 recognised 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.

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 compatability.

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.

See Also:

- Implemented as `gdal/frmts/hfa/hfadataset.cpp`.
- More information, and other tools are available on the Imagine (.img) Reader page.
- Erdas.com

RST --- Idrisi Raster Format

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 interleafed 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 companying 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 companying 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

See Also:

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

INGR --- Intergraph Raster Format

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.

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 CYMK
- 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 untilled instance of any of the supported data formats.

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

Note that writing in that format is not encouraged.

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.

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 companying .dgn file or in application specific data storage inside the file itself.

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/

ISIS2 -- USGS Astrogeology ISIS Cube (Version 2)

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.

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.

Creation Options

- **LABELING_METHOD=ATTACHED/DETACHED**: Determines whether the header labelling 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.

See Also:

- Implemented as `gdal/frmts/pds/isis2dataset.cpp`.
- GDAL PDS Driver
- GDAL ISIS3 Driver

ISIS3 -- USGS Astrogeology ISIS Cube (Version 3)

ISIS3 is a format used by the USGS Planetary Cartography group to store and distribute planetary imagery data. GDAL provides read-only 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.

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.

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.

See Also:

- Implemented as `gdal/frmts/pds/isis2dataset.cpp`.
- GDAL PDS Driver
- GDAL ISIS2 Driver

JP2ECW -- ERDAS JPEG2000 (.jp2)

GDAL supports reading and writing JPEG2000 files using the ECW SDK from ERDAS.

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.

Support for the JP2ECW driver in GDAL is optional, and requires linking in external ECW SDK libraries provided by ERDAS.

Creation Issues

The ECW 4.x SDK from ERDAS 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.

Creation Options:

- **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.
- **LARGE_OK=YES**: When built with the ECW 3.x SDK this option can be set to allow compressing files larger than 500MB. It is the users responsibility to ensure that the licensing requirments for large file compression are being adhered to.
- **ECW_ENCODE_KEY=key**: Provide the OEM encoding key purchased from Erdas which permits encoding images. The key is is approximately 129 hex digits long. It may also be provided globally as a configuration option.
- **ECW_ENCODE_COMPANY=name**: Provide the name of the company ERDAS issued the OEM encoding key (see ECW_ENCODE_KEY) to. This must exactly match the name used by ERDAS in issuing the OEM key. It may also be provided globally as a configuration option.
- **GMLJP2=YES/NO**: Indicates whether a GML box conforming to the OGC GML in JPEG2000 specification should be included in the file. Defaults to YES.
- **GeoJP2=YES/NO**: Indicates whether a GML 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.
- **CODESTREAM_ONLY=YES/NO**: If set to YES, only the compressed imagery code stream will be written. If NO (the default) a JP2 package will be written around the code stream including a variety of meta information.
- **LEVELS=n**: See ECW SDK for details.
- **LAYERS=n**: See ECW SDK for details.
- **PRECINCT_WIDTH=n**: See ECW SDK for details.
- **PRECINCT_HEIGHT=n**: See ECW SDK for details.
- **TILE_WIDTH=n**: See ECW SDK for details.
- **TILE_HEIGHT=n**: See ECW SDK for details.
- **INCLUDE_SOP=YES/NO**: See ECW SDK for details.

- **INCLUDE_EPH=YES/NO**: See ECW SDK for details.
- **DECOMPRESS_LAYERS=n**: See ECW SDK for details.
- **DECOMPRESS_RECONSTRUCTION_PARAMETER=n**: See ECW SDK for details.

JPEG2000 format does not support creation of overviews since the format is already considered to be optimized for "arbitrary overviews".

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.
- **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.
- **ECW_ENCODE_KEY, ECW_ENCODE_COMPANY**: These values, as described in the creation options, may also be set as configuration options. See above.

See Also

- Implemented as `gdal/frmts/ecw/ecwdataset.cpp`.
- ECW SDK available at erdas.com.
- GDAL ECW Build Hints

JP2KAK -- JPEG-2000 (based on Kakadu)

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 commercial 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. GDAL includes another JPEG2000 driver based on the free JasPer library.

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.

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:

- **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.
- **GMLJP2=YES/NO**: Indicates whether a GML box conforming to the OGC GML in JPEG2000 specification should be included in the file. Defaults to YES.
- **GeoJP2=YES/NO**: Indicates whether a GML 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.

See Also:

- Implemented as `gdal/frmts/jp2kak/jp2kakdataset.cpp`.
- JPEG2000 for Geospatial Applications page, includes GeoJP2(tm) discussion.
- Alternate JPEG200 driver.

JP2MrSID --- JPEG2000 via 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.

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 a LizardTech-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>LizardTech 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
    </wavelet97>
    <precinctSize>
      256 256
    </precinctSize>
  </Codestream>
</Jp2Profile>
```

See Also:

- Implemented as `gdal/frmts/mrsid/mrsiddataset.cpp`.
- LizardTech's Web site

JP2OpenJPEG --- JPEG2000 driver based on OpenJPEG library

(GDAL >= 1.8.0)

This driver is an implementation of a JPEG2000 reader/writer based on OpenJPEG library **v2**.

The v2 branch is - at the time of writing - still unreleased, so you'll have to pull it yourself from its Subversion repository : http://openjpeg.googlecode.com/svn/branches/v2. OpenJPEG trunk is still deriving on the 1.3 series that doesn't provide access to tile-level reading. v2 branch enables reading big JPEG2000 images without loading them entirely in memory. This is a noticeable improvement in comparison with the JPEG2000 driver based on Jasper library.

The v2 branch also adds the capability to use the VSI Virtual File API, so the driver is able to read JPEG2000 compressed NITF files.

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.

In creation, the driver doesn't support writing GeoJP2 nor GMLJP2.

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's neither JP2 nor J2K, J2K codec is used.
- **QUALITY** : 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
- **REVERSIBLE=YES/NO** : YES means lossless compression. Defaults to NO.
- **RESOLUTIONS** : Number of resolution levels. Between 1 and 7. Defaults to 6.
- **BLOCKXSIZE** : Tile width. Defaults to 1024.
- **BLOCKYSIZE** : Tile height. Defaults to 1024.
- **PROGRESSION** : Progession order : LRCP, RLCP, RPCL, PCRL or CPRL. Defaults to LRCP.
- **SOP=YES/NO** : YES means generate SOP marker segments. Defaults to NO.
- **EPH=YES/NO** : YES means generate EPH marker segments. Defaults to NO.

Patches for OpenJPEG library :

Links to usefull patches to apply to OpenJPEG (valid for v2 branch in revision r565) :

- Fix leak in jp2_read_header_procedure()
- Fix internal function names conflict with Jasper that can cause crashes

See Also:

- Implemented as `gdal/frmts/openjpeg/openjpegdataset.cpp`.
- Official JPEG-2000 page
- The OpenJPEG library home page

Other JPEG2000 GDAL drivers :

- JPEG2000: based on Jasper library (open source)
- JP2ECW: based on Erdas ECW library (commercial)

- JP2MRSID: based on LizardTech MrSID library (commercial)
- JP2KAK: based on Kakadu library (commercial)

JPEG2000 --- Implementation of the JPEG-2000 part 1

This implementation based on JasPer software (see below).

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.

Creation Options

- **WORLDFILE=ON**: Force the generation of an associated ESRI world file (.wld).
- **FORMAT=JP2|JPC**: Specify output file format.
- 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:

imgareatlx=x	Set the x-coordinate of the top-left corner of the image area to x.
imgareatly=y	Set the y-coordinate of the top-left corner of the image area to y.
tilegrdtlx=x	Set the x-coordinate of the top-left corner of the tiling grid to x.
tilegrdtly=y	Set the y-coordinate of the top-left corner of the tiling grid to y.
tilewidth=w	Set the nominal tile width to w.
tileheight=h	Set the nominal tile height to h.
prcwidth=w	Set the precinct width to w. The argument w must be an integer power of two. The default value is 32768.
prcheight=h	Set the precinct height to h. The argument h must be an integer power of two. The default value is 32768.
cblkwidth=w	Set the nominal code block width to w. The argument w must be an integer power of two. The default value is 64.
cblkheight=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: (see table below) 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.
ilyrrates=[, ,.. . ,]	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).

Value	Description
int	integer mode
real	real mode

prg=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-component-position (RLCP) progressive (i.e., resolution scalable)
rpcl	resolution-position-component-layer (RPCL) progressive
pcrl	position-component-resolution-layer (PCRL) progressive
cprl	component-position-resolution-layer (CPRL) progressive

By default, LRCP progressive ordering is employed. Note that the RPCL and PCRL progressions are not valid for all possible image geometries. (See standard for more details.)

nomct	Disallow the use of any multicomponent transform.
numrlvls=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).
termall	Terminate all coding passes.
segsym	Use segmentation symbols.
vcausal	Use vertically stripe causal contexts.
pterm	Use predictable termination.
resetprob	Reset the probability models after each coding pass.
numgbits=n	Set the number of guard bits to n."

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 ftp://ftp.remotesensing.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 (commercial)
- JP2MRSID: based on LizardTech MrSID library (commercial)
- JP2KAK: based on Kakadu library (commercial)

JPEG -- JPEG JFIF File Format

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 metedata of the IMAGE_STRUCTURE domain.

There is currently no support for georeferencing information or metadata for JPEG files. But if an ESRI world file exists with the .jgw, .jpgw/.jpegw or .wld suffixes, it will be read and used to establish the geotransform for the image. If available a MapInfo .tab file will also be used for georeferencing. Overviews can be built for JPEG files as an external .ovr file.

The driver also supports the "zlib compressed mask appended to the file" approach used by a few data providers to add a bitmask to identify pixels that are not valid data. See RFC 15 for further details.

The 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 explicitely 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.

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 colorspaces 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 but 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.

See Also:

- Independent JPEG Group
- libjpeg-turbo
- GDAL GeoTIFF driver

JPEGLS

(GDAL >= 1.8.0)

This driver is an implementation of a JPEG-LS reader/writer based on the Open Source CharLS library (BSD style license). At the time of writing, the CharLS library in its 1.0 version has no "make install" target. Consequently, the integration of the driver in the GDAL build system is a big rough. On Unix, you have to edit the GDALmake.opt file and edit the lines related to CHARLS.

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.

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.

See Also:

- Implemented as `gdal/frmts/jpegls/jpeglsdataset.cpp`.
- Homepage of the CharLS library

JPIPKAK - JPIP Streaming

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.

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.

JPIP Sequence Diagram

1. Initial JPIP request for a target image, a target id, a session over http, data to be returned as a jpp-stream are requested and a maximum length is put on the response. In this case no initial window is requested, though it can be. Server responds with a target identifier that can be used to identify the image on the server and a JPIP-cnew response header which includes the path to the JPIP server which will handle all future requests and a cid session identifier. A session is required so that that the server can model the state of the client connection, only sending the data that is required.
2. Client requests particular view windows on the target image with a maximum response length and includes the session identifier established in the previous communication. 'fsiz' is used to identify the resolution associated with the requested view-window. The values 'fx' and 'fy' specify the dimensions of the desired image resolution. 'roff' is used to identify the upper left hand corner off the spatial region associated with the requested view-windw. 'rsiz' is used to identify the horizontal and vertical extents of the spatial region associated with the requested view-window.

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.

JPIPKAK - implementation

The implementation supports the GDAL C++ and C API, and provides an initial SWIG wrapper for this driver with a Java ImageIO example (**TODO** - qGIS Example).

The driver uses a simple threading model to support requesting reads of the data and remote fetching. This threading model supports two separate client windows, with just one connection to the server. Requests to the server are multiplexed to utilize available bandwidth efficiently. The client identifies these windows by using 0 (low) or 1 (high) values to a PRIORITY metadata request option.

Note: SSL support

If the client is built with support for SSL, then driver determines whether to use SSL if the request is a jpips:// protocol as opposed to jpip:// . Note that the driver does not verify server certificates using the Curl certificate bundle and is currently set to accept all SSL server certificates.

Note: libCurl

JPIP sets client/server values using HTTP headers, modifications have been made to the GDAL HTTP portability library to support this.

GDAL Sequence Diagram

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 initialised, otherwise NULL is returned.
3. Initialize

   Initialisation involves making an initial connection to the JPIP Server to establish a session and to retrieve the initial metadata about the image (ref. JPIP Sequence Diagram).

   If the connection fails, the function returns false and the `Open` function returns NULL indicating that opening the dataset with this driver failed.

   If the connection is successful, then subsequent requests to the JPIP server are made to retrieve all the available metadata about the image. Metadata items are set using the `GDALMajorObject->SetMetadataItem` in the "JPIP" domain.

   If the metadata returned from the server includes GeoJP2 UUID box, or a GMLJP2 XML box then this metadata is parsed and sets the geographic metadata of this dataset.
4. GDALGetMetadata

   C API to `JPIPKAKDataset->GetMetadata`
5. GetMetadata

   returns metadata for the "JPIP" domain, keys are "JPIP_NQUALITYLAYERS", "JPIP_NRESOLUTIONLEVELS", "JPIP_NCOMPS" and "JPIP_SPRECISION"
6. GDALEndAsyncRasterIO

   If the asynchronous raster IO is active and not required, the C API calls `JPIPKAKDataset->EndAsyncRasterIO`
7. EndAsyncRasterIO

   The JPIPKAKAsyncRasterIO object is deleted
8. delete
9. GDALBeginAsyncRasterIO

C API to `JPIPKAKDataset->BeginAsyncRasterIO`

10. BeginAsyncRasterIO

    The client has set the requested view window at 1:1 and have optionally set the discard level, quality layers and thread priority metadata items.
11. Create

    Creates a JPIPKAKAsyncRasterIO Object
12. Start

    Configures the kakadu machinery and starts a background thread (if not already running) to communicate to the server the current view window request. The background thread results in the `kdu_cache` object being updated until the JPIP server sends an "End Of Response" (EOR) message for the current view window request.
13. GDALLockBuffer

    C API to LockBuffer
14. LockBuffer

    Not implemented in `JPIPKAKAsyncRasterIO`, a lock is acquired in `JPIPKAKAsyncRasterIO->GetNextUpdatedRegion`
15. GDALGetNextUpdatedRegion

    C API to GetNextUpdatedRegion
16. GetNextUpdatedRegion

    The function decompresses the available data to generate an image (according to the dataset buffer type set in `JPIPKAKDataset->BeginAsyncRasterIO`) The window width, height (at the requested discard level) decompressed is returned in the region pointer and can be rendered by the client. The status of the rendering operation is one of `GARIO_PENDING, GARIO_UPDATE, GARIO_ERROR, GARIO_COMPLETE` from the `GDALAsyncStatusType` structure. `GARIO_UPDATE, GARIO_PENDING` require more reads of GetNextUpdatedRegion to get the full image data, this is the progressive rendering of JPIP. `GARIO_COMPLETE` indicates the window is complete.

    `GDALAsyncStatusType` is a structure used by`GetNextUpdatedRegion` to indicate whether the function should be called again when either kakadu has more data in its cache to decompress, or the server has not sent an End Of Response (EOR) message to indicate the request window is complete.

    The region passed into this function is passed by reference, and the caller can read this region when the result returns to find the region that has been decompressed. The image data is packed into the buffer, e.g. RGB if the region requested has 3 components.
17. GDALUnlockBuffer

    C Api to UnlockBuffer
18. UnlockBuffer

    Not implemented in `JPIPKAKAsyncRasterIO`, a lock is acquired in `JPIPKAKAsyncRasterIO->GetNextUpdatedRegion`
19. Draw

    Client renders image data
20. GDALLockBuffer
21. LockBuffer
22. GDALGetNextUpdatedRegion
23. GetNextUpdatedRegion

24. GDALUnlockBuffer
25. UnlockBuffer
26. Draw

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.

See Also:

- JPEG 2000 Interactive Protocol (Part 9 JPIP)
- http://www.opengeospatial.org/standards/gmljp2
- Kakadu Software
- IAS demo (example JPIP(S) streams)

NOTES

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

L1B -- NOAA Polar Orbiter Level 1b Data Set (AVHRR)

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.

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.
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.
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.

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.

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.

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
- 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

GDAL Driver for FARSITE v.4 LCP Format

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 USGS National Map for LANDFIRE distributes data in LCP format, and programs such as FARSITE and LFDAT can create LCP files from a set of input rasters. The GDAL driver for LCP supports reading only.

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.

The GDAL LCP driver reports dataset- and band-level metadata:

Dataset

LATITUDE: Latitude of the dataset, negative for southern hemisphere
LINEAR_UNIT: Feet or meters
DESCRIPTION: LCP file description

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.

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, canopy base height, canopy bulk density)
4	4	long	ground fuels	20 if no ground fuels, 21 if ground fuels exist (ground fuels = duff loading, coarse woody)
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
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 models BUT no conversion file, 2 = no custom models BUT conversion file, 3 = custom models AND conversion file needed
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 1000
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

Leveller --- Daylon Leveller Heightfield

Leveller heightfields store 32-bit elevation values. Format versions 4 through 7 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 autoconverts them to floating-point.

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 version 7, it returns a realworld 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 realworld) transform for the elevation data.

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 (realworld) elevations to physical elevations.

Header information is written out on the first call to `band::IWriteBlock`.

History

v1.2 (Jul 17/07): Added v7 and Create support.

v1.1 (Oct 20/05): Fixed coordsys and elev scaling errors.

See Also:

- Implemented as `gdal/frmts/leveller/levellerdataset.cpp`.
- Leveller SDK, which documents the Leveller format.

MEM -- In Memory Raster

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 auxilary information including metadata, coordinate systems, georeferencing, GCPs, color interpretation, nodata, color tables and all pixel data types.

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
```

- DATAPOINTER: pointer to the first pixel of the first band represented as a long integer. NOTE! This may not work on platforms where a long is 32 bits and a pointer is 64 bits. (required)
- 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.

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.

MFF2 -- Vexcel MFF2 Image

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`.

Format Details

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.

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 }
```

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.

The "georef" File

The "georef" file is used to describe the geocoding and projection information for the binary data. For example,

```
top_left.latitude            = 32.93333333333334
top_left.longitude           = 130.0
top_right.latitude           = 32.93333333333334
top_right.longitude          = 130.5
bottom_left.latitude         = 32.50000000000001
bottom_left.longitude        = 130.0
bottom_right.latitude        = 32.50000000000001
bottom_right.longitude       = 130.5
centre.latitude              = 32.71666666666668
centre.longitude             = 130.25
projection.origin_longitude  = 0
projection.name              = ll
spheroid.name                = wgs-84
```

describes an orthogonal latitude/longitude (ll) projected image, with latitudes and longitudes based on the wgs-84 ellipsoid.

Since MFF2 version 1.1, top_left refers to the top left corner of the top left pixel. top_right refers to the top right corner of the top right pixel. bottom_left refers to the bottom left corner of the bottom left pixel. bottom_right refers to the bottom right corner of the bottom right pixel. centre refers to the centre of the four corners defined above (center of the image).

Mathematically, for an Npix by Nline image, the corners and centre in (pixel,line) coordinates for MFF2 version 1.1 are:

```
top_left: (0,0)
top_right: (Npix,0)
bottom_left: (0,Nline)
bottom_right: (Npix,Nline)
centre: (Npix/2.0,Nline/2.0)
```

These calculations are done using floating point arithmetic (ie. centre coordinates may take on non-integer values).

Note that the corners are always expressed in latitudes/longitudes, even for projected images.

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, ie. 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.

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
```

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.
```

MrSID --- Multi-resolution Seamless Image Database

MrSID is a wavelet-based image compression technology which can utilise 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 is is developed and distributed by the LizardTech, Inc.

This driver supports reading of MrSID image files using LizardTech's decoding software development kit (DSDK). **This DSDK is not free software, you should contact LizardTech 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.

Metadata

MrSID metadata transparently translated into GDAL metadata strings. Files in MrSID format contain a set of standard metadata tags such as: IMAGE__WIDTH (contains the width of the image), IMAGE__HEIGHT (contains the height of the image), IMAGE__XY_ORIGIN (contains the x and y coordinates of the origin), IMAGE__INPUT_NAME (contains the name or names of the files used to create the MrSID image) etc. GDAL's metadata keys cannot contain characters `:' and `=', but standard MrSID tags always contain double colons in tag names. These characters replaced in GDAL with `_' during translation. So if you are using other software to work with MrSID be ready that names of metadata keys will be shown differently in GDAL.

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.

Georeference

MrSID images may contain georeference and coordinate system information in form of GeoTIFF GeoKeys, translated in metadata records. All those GeoKeys properly extracted and used by the driver. Unfortunately, there is one caveat: old MrSID encoders has a bug which resulted in wrong GeoKeys, stored in MrSID files. This bug was fixed in MrSID software version 1.5, but if you have older encoders or files, created with older encoders, you cannot use georeference information from them.

Creation Options

MrSID writing is only supported if GDAL is built against a MrSID ESDK 5.x or newer. This is normally only licensed by Lizardtech under controlled circumstances (whereas the DSDK is free for anyone to use within the constraints of a license agreement). If you do have the ESDK, it can be used to write MrSID files. The following creation options are supported.

- WORLDFILE: Yes to write an ESRI world file (with the extension .sdw).
- VERSION: Can be 2 for MrSID version 2 files, or 3 for MrSID version 3 files.
- COMPRESSION: Used only for version 2 MrSID files. Indicates the desired compression ratio. MrSID generation 2 cannot compress an image losslessly; so zero does *not* indicate numerically lossless compression. A value of one may be used for highest fidelity, but a value of twenty will generally yield the best results (a "visually lossless" compression). Twenty would indicate a 20:1 compression ratio (the image would be compressed to 1/20 its original size).
- FILESIZE: Used only for version 3 MrSID files. Indicates the desired output file size in bytes. Use "0" for lossless compression.

- TWOPASS: Used only with 3 MrSID files. Indicates that a two pass optimizer algorithm should be used during compression.

See Also:

- Implemented as `gdal/frmts/mrsid/mrsiddataset.cpp`.
- LizardTech's Web site

MrSID/MG4 LiDAR Compression / Point Cloud View files

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 LizardTech. This driver supports reading of MG4 LiDAR files using LizardTech's decoding software development kit (DSDK). **This DSDK is freely distributed; but, it is not open source software. You should contact LizardTech to obtain it (see link at end of this page).**

Example View files (from View Document specification)

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 NOFITLER 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 ClassificationFitler 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>
```

Writing not supported

This driver does not support writing MG4 files.

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.

See Also:

- Implemented as *gdal/frmts/mrsid_lidar/gdal_MG4Lidar.cpp*
- MrSID/MG4 LiDAR View Document Specification
- LizardTech's Web site

MSG -- Meteosat Second Generation

This driver implemets 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 a licensed software and available free to download upon acceptance of the WaveLet Transform Software Licence during electronic registration process.

Part of the source of the file xrithdr_extr.c from XRIT2PIC is used to parse MSG headers. This source is licensed under the terms of the GNU General Public License as published by the Free Software Foundation.

This driver is not "enabled" by default. See Build Instructions on how to include this driver in your GDAL library.

Build Instructions

Download the Eumetsat library for wavelet decompression. This is a file named PublicDecompWT.zip. Extract the content in a subdirectory with the same name (under frmts/msg).

If you are building with Visual Studio 6.0, extract the .vc makefiles for the PublicDecompWT from the file PublicDecompWTMakefiles.zip.

If you build using the GNUMakefile, use *--with-msg* option to enable MSG driver:

```
./configure --with-msg
```

If 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 indepentent", 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 GDAL 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

Specification of Source Dataset

It is possible to select individual files for opening. In this case, the driver will gather the files that correspond to the other strips of the same image, and correctly compose the image.

Example with gdal_translate.exe:

```
gdal_translate
 C:\hrit_a\2004\05\31\H-000-MSG1__-MSG1________-HRV______-000008___-200405311115-C_
 C:\output\myimage.tif
```

It is also possible to use the following syntax for opening the MSG files:

- MSG(source_folder,timestamp,(channel,channel,...,channel),use_root_folder,data_conversion,nr_cycles,step)
-
    - source_folder: a path to a folder structure that contains the files

- timestamp: 12 digits representing a date/time that identifies the 114 files of the 12 images of that time, e.g. 200501181200
- channel: a number between 1 and 12, representing each of the 12 available channels. When only specifying one channel, the brackets are optional.
- use_root_folder: Y to indicate that the files reside directly into the source_folder specified. N to indicate that the files reside in date structured folders: source_folder/YYYY/MM/DD
- data_conversion:
- ◊ N to keep the original 10 bits DN values. The result is UInt16.
  ◊ B to convert to 8 bits (handy for GIF and JPEG images). The result is Byte.
  ◊ R to perform radiometric calibration and get the result in mW/m2/sr/(cm-1)-1. The result is Float32.
  ◊ L to perform radiometric calibration and get the result in W/m2/sr/um. The result is Float32.
  ◊ T to get the reflectance for the visible bands (1, 2, 3 and 12) and the temprature in degrees Kelvin for the infrared bands (all other bands). The result is Float32.
- nr_cycles: a number that indicates the number of consecutive cycles to be included in the same file (time series). These are appended as additional bands.
- step: a number that indicates what is the stepsize when multiple cycles are chosen. E.g. every 15 minutes: step = 1, every 30 minutes: step = 2 etc. Note that the cycles are exactly 15 minutes apart, so you can not get images from times in-between (the step is an integer).

Examples with gdal_translate utility:

Example call to fetch an MSG image of 200501181200 with bands 1, 2 and 3 in IMG format:

```
gdal_translate -of HFA MSG(\\pc2133-24002\RawData\,200501181200,(1,2,3),N,N,1,1)
   d:\output\outfile.img
```

In JPG format, and converting the 10 bits image to 8 bits by dividing all values by 4:

```
gdal_translate -of JPEG MSG(\\pc2133-24002\RawData\,200501181200,(1,2,3),N,B,1,1)
   d:\output\outfile.jpg
```

The same, but reordering the bands in the JPEG image to resemble RGB:

```
gdal_translate -of JPEG MSG(\\pc2133-24002\RawData\,200501181200,(3,2,1),N,B,1,1)
   d:\output\outfile.jpg
```

Geotiff output, only band 2, original 10 bits values:

```
gdal_translate -of GTiff MSG(\\pc2133-24002\RawData\,200501181200,2,N,N,1,1)
   d:\output\outfile.tif
```

Band 12:

```
gdal_translate -of GTiff MSG(\\pc2133-24002\RawData\,200501181200,12,N,N,1,1)
   d:\output\outfile.tif
```

The same band 12 with radiometric calibration in mW/m2/sr/(cm-1)-1:

```
gdal_translate -of GTiff MSG(\\pc2133-24002\RawData\,200501181200,12,N,R,1,1)
   d:\output\outfile.tif
```

Retrieve data from c:\hrit-data\2005\01\18 instead of \\pc2133-24002\RawData\... :

```
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
```

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.

See Also:

- Implemented as `gdal/frmts/msg/msgdataset.cpp`.
- http://www.eumetsat.int - European Organisation for the Exploitation of Meteorological Satellites

MSGN -- Meteosat Second Generation (MSG) Native Archive Format (.nat)

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.

NetCDF: Network Common Data Form

This format is supported for read and write access. NetCDF is an interface for array-oriented data access and is used for representing scientific data.

The fill value metadata or missing_value backward compatibililty is preserved as NODATA value when available.

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

Multiple Image Handling (Subdatasets)

Nework 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 withing 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 O1 (13 November 2004)
  NC_GLOBAL#experiment_id=SRES A2 experiment
  NC_GLOBAL#realization=1
  NC_GLOBAL#cmor_version=9.600000e-01
  NC_GLOBAL#Conventions=CF-1.0
  NC_GLOBAL#history=YYYY/MM/JJ: data generated; YYYY/MM/JJ+1 data transformed  At 16:37:23
on 01/11/2005, CMOR rewrote data to comply with CF standards and IPCC Fourth Assessment
requirements
  NC_GLOBAL#references=Dufresne et al, Journal of Climate, 2015, vol XX, p 136
  NC_GLOBAL#comment=Test drive
Subdatasets:
  SUBDATASET_1_NAME=NETCDF:"sst.nc":lon_bnds
  SUBDATASET_1_DESC=[180x2] lon_bnds (64-bit floating-point)
  SUBDATASET_2_NAME=NETCDF:"sst.nc":lat_bnds
  SUBDATASET_2_DESC=[170x2] lat_bnds (64-bit floating-point)
  SUBDATASET_3_NAME=NETCDF:"sst.nc":time_bnds
  SUBDATASET_3_DESC=[24x2] time_bnds (64-bit floating-point)
  SUBDATASET_4_NAME=NETCDF:"sst.nc":tos
  SUBDATASET_4_DESC=[24x170x180] sea_surface_temperature (32-bit floating-point)Corner
Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0,  512.0)
Upper Right (  512.0,    0.0)
Lower Right (  512.0,  512.0)
Center      (  256.0,  256.0)
```

This netCDF files contain 4 datasets, lon_bnds, lat_bnds, tim_bnds and tos. Now select the subdataset, described as:

NETCDF:"sst.nc":tos

[24x17x180] sea_surface_temperature (32-bit floating-point)

and get the information about the number of bands there is inside this variable.

```
$ gdalinfo NETCDF:"sst.nc":tos
Driver: netCDF/Network Common Data Format
Size is 180, 170
Coordinate System is `'
Origin = (1.000000,-79.500000)
Pixel Size = (1.98888889,0.99411765)
Metadata:
  NC_GLOBAL#title=IPSL  model output prepared for IPCC Fourth Assessment SRES A2 experiment
  NC_GLOBAL#institution=IPSL (Institut Pierre Simon Laplace, Paris, France)
```

.... More metadata

```
  time#standard_name=time
  time#long_name=time
  time#units=days since 2001-1-1
  time#axis=T
  time#calendar=360_day
  time#bounds=time_bnds
  time#original_units=seconds since 2001-1-1
Corner Coordinates:
Upper Left  (   1.0000000, -79.5000000)
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 display 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.

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.

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 mehtods fail, NetCDF driver will try to read the following metadata directly and set up georefenrencing.

- spatial_ref (Well Known Text)

- GeoTransform (GeoTransform array)

or,

- Northernmost_Northing
- Southernmost_Northing
- Easternmost_Easting
- Westernmost_Easting

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.

GDAL NetCDF Metadata

All netCDF attributes are transparently translated as GDAL metadata.

The translation follow these directives:

- Global NetCDF metatadata 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
```

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.

Please note that with **CygWIN** you need to be sure that the DLLs are executable, or GDAL will not execute.

```
chmod a+rx [NetCDF DLLs]
```

The netCDF DLLs directory **must be** to be in your **PATH**.

See Also:

- NetCDF CF-1.0 convention
- NetCDF compiled libraries
- NetCDF Documentation

Full list of GDAL Raster Formats

NITF -- National Imagery Transmission Format

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 auxilary 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.

Most file header and image header fields are returned as dataset level metadata.

Creation Issues

On export NITF files are always written as NITF 2.1 with one image and no other auxilary 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 methology 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 missle 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 driver documentation. Starting with GDAL 1.7.0, 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 both CreateCopy() and Create() methods. JPEG2000 compression is only available if the JP2ECW driver is available. The TARGET and PROFILE JP2ECW-specific creation options can be used. See the JP2ECW driver documentation. Starting with GDAL 1.7.0, if JP2ECW driver is not available, Jasper JPEG2000 driver can be used 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()).
- **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 blackslashes 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.

Links

- Advanced GDAL NITF Driver Information
- NITFS Technical Board Public Page
- DIGEST Part 2 Annex D (describe encoding of NITF Spatial Data Extensions)
- RPFTOC driver : to read the Table Of Contents of CIB and CADRG products.
- MIL-PRF-89038 : specification of RPF, CADRG, CIB products
- ECRGTOC driver : to read the Table Of Contents of ECRG products.
- MIL-PRF-32283 : specification of ECRG products

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.

OGDI -- OGDI Bridge

Note : From GDAL >= 1.5.0, there should be little reason to use the OGDI raster bridge, as ADRG, DTED and RPF (CADRG/CIB) formats are natively supported by GDAL.

OGDI raster data sources are supported by GDAL for reading. Both Matrix and Image families should be fully supported, as well as reading of colormap and projection metadata. The GDAL reader is intended to be used with OGDI 3.1 drivers, but OGDI 3.0 drivers should also work.

OGDI datasets are opened within GDAL by selecting the GLTP url. For instance, gltp://gdal.velocet.ca/adrg/usr4/mpp1/adrg/TPSUS0101 would open the ADRG dataset stored at /usr4/mpp1/adrg/TPSUS0101 on the machine gdal.velocet.ca (assuming it has an OGDI server running) using the 'adrg' driver. This default access to a whole datastore will attempt to represent all layers (and all supported family types) as bands all at the resolution and region reported by the datastore when initially accessed.

It is also possible to select a particular layer and access family from an OGDI datastore by indicating the layer name family in the name. The GDAL dataset name gltp:/adrg/usr4/mpp1/adrg/TPUS0101:"TPUS0102.IMG":Matrix would select the layer named TPUS0102.IMG from the dataset /usr4/mpp1/adrg/TPUS0101 on the local system using the ADRG driver, and access the Matrix family. When a specific layer is accessed in this manner GDAL will attempt to determine the region and resolution from the OGDI 3.1 capabilities document. Note that OGDI 3.0 datastores must have the layer and family specified in the dataset name since they cannot be determined automatically.

```
eg.
  gltp://gdal.velocet.ca/adrg/usr4/mpp1/adrg/TPUS0101
  gltp:/adrg/usr4/mpp1/adrg/TPUS0101
  gltp:/adrg/usr4/mpp1/adrg/TPUS0101:"TPUS0102.IMG":Matrix
```

OGDI Matrix family layers (pseudocolored integer layers) are represented as a single band of raster data with a color table. Though the Matrix layers contain 32bit integer values, they are represented through GDAL as eight layers. All values over 255 are truncated to 255, and only 256 colormap entries are captured. While this works well for most Matrix layers, it is anticipated that at some point in the future Matrix layers with a larger dynamic range will be represented as other data types.

OGDI Image family layers may internally have a type of RGB (1) which is represented as three 8bit bands in GDAL, or Byte (2), UInt16 (3), Int16 (4) or Int32 (5). There is no support for floating points bands in OGDI 3.1.

The GDAL OGDI driver will represent OGDI datasources as having *arbitrary* overviews. Any GDAL raster read requests at a reduced resolution will be passed on to the OGDI driver at that reduced resolution; potentially allowing efficient reading of overview information from OGDI datastores.

If an OGDI datastore is opened without selecting a layer name in the dataset name, and if the datastore has OGDI 3.1 style capabilities, the list of layers will be made available as SUBDATASETS metadata. For instance, the *gdalinfo* command might report the following. This information can be used to establish available layers for direct access.

```
Subdatasets:
  SUBDATASET_1_NAME=gltp:/adrg/usr4/mpp1/adrg/TPUS0101:"TPUS0101.IMG":Matrix
  SUBDATASET_1_DESC=TPUS0101.IMG as Matrix
  SUBDATASET_2_NAME=gltp:/adrg/usr4/mpp1/adrg/TPUS0101:"TPUS0102.IMG":Matrix
  SUBDATASET_2_DESC=TPUS0102.IMG as Matrix
  SUBDATASET_3_NAME=gltp:/adrg/usr4/mpp1/adrg/TPUS0101:"TPUS0101.IMG":Image
  SUBDATASET_3_DESC=TPUS0101.IMG as Image
  SUBDATASET_4_NAME=gltp:/adrg/usr4/mpp1/adrg/TPUS0101:"TPUS0102.IMG":Image
  SUBDATASET_4_DESC=TPUS0102.IMG as Image
```

See Also:

- ogdi.sourceforge.net

OZI -- OZF2/OZFX3 raster

(Available for GDAL >= 1.8.0)

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.

See also

- Specification of OZF2/OZFX3 format

JAXA PALSAR Processed Products

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.

See Also:

- RESTEC Sample Data

PCIDSK --- PCI Geomatics Database File

PCIDSK database file used by PCI EASI/PACE software for image analysis. It is supported for reading, and writing by GDAL. All pixel data types, and data organizations (pixel interleaved, band interleaved, file interleaved and tiled) should be supported. Currently LUT segments are ignored, but PCT segments should be treated as associated with the bands. Overall file, and band specific metadata should be correctly associated with the image or bands.

Georeferencing is supported though there may be some limitations in support of datums and ellipsoids. GCP segments are ignored. RPC segments will be returned as GDAL style RPC metadata.

Internal overview (pyramid) images will also be correctly read though newly requested overviews will be built externally as an .ovr file.

Vector segments are supported by the OGR PCIDSK driver.

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.

See Also:

- Implemented as `gdal/frmts/pcidsk/pcidskdataset2.cpp`.
- PCIDSK SDK

Geospatial PDF

(Available for GDAL >= 1.8.0)

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.

GDAL must be compiled with libpoppler support (GPL-licensed), and 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. Successfully tested versions are poppler >= 0.12.X and <= 0.16.0.

Starting with GDAL 1.9.0, as an alternative, the PDF driver can be compiled against libpodofo (LGPL-licensed) to avoid the libpoppler dependency. This is sufficient to get the georeferencing 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 and 0.9.1.

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.

The dimensions of the raster can be controlled by specifying the DPI of the rasterization with the *GDAL_PDF_DPI* configuration option. Its default value is 150.

Multipage documents are exposed as subdatasets, one subdataset par page of the document.

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.

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, which can be a slow operation.

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).

There is currently no support for selective layer rendering.

See also

- OGC GeoPDF Encoding Best Practice Version 2.2
- Adobe Supplement to ISO 32000
- Poppler homepage
- A few Geospatial PDF samples
- Another set of Geospatial PDF samples

PDS -- Planetary Data System

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.

Due to abiguitities 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 #3940.

PDS is part of a family of related formats including ISIS2 and ISIS3.

See Also:

- Implemented as `gdal/frmts/pds/pdsdataset.cpp`.
- NASA Planetary Data System
- GDAL ISIS2 Driver
- GDAL ISIS3 Driver

Rasdaman GDAL driver

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 to deliver 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.

See Also:

- [Rasdaman Project](http://www.rasdaman.org)

Rasterlite - Rasters in SQLite DB

Starting with GDAL 1.7.0, 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 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.

Connexion 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=ma

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 correspondings 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.

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 exists. In that case, the raster table name will be base on the DB name itself.

Creation options

- **WIPE** (=NO by default): Set to YES to erase all prexisting 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'.

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).

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 = (-5.000000000000000,55.000000000000000)
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
  ```

See Also:

- Spatialite and Rasterlite home page
- Rasterlite manual
- Rasterlite howto
- Sample databases

R -- R Object Data Store

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.

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

RIK -- Swedish Grid Maps

Supported by GDAL for read access. This format is used in maps issued by the swedish organization Lantmteriet. 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`.

RMF --- Raster Matrix Format

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 flavours 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.

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.

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.

See Also:

- Implemented as `gdal/frmts/rmf/rmfdataset.cpp`.
- "Panorama" GIS homepage

RS2 -- RadarSat 2 XML Product

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.

Data Calibration

If you wish to have GDAL apply a particular calibration LUT to the data when you open it, you have to open the appropriate subdatasets. The following subdatasets exist within the SUBDATASET domain for RS2 products:

- uncalibrated - open with RADARSAT_2_CALIB:UNCALIB: prepended to filename
- $beta_0$ - open with RADARSAT_2_CALIB:BETA0: prepended to filename
- $sigma_0$ - open with RADARSAT_2_CALIB:SIGMA0: prepended to filename
- gamma - open with RADARSAT_2_CALIB:GAMMA: prepended to filename

Note that geocoded (SPG/SSG) products do not have this functionality available. Also be aware that the LUTs must be in the product directory where specified in the product.xml, otherwise loading the product with the calibration LUT applied will fail.

One caveat worth noting is that the RADARSAT-2 driver will supply the calibrated data as GDT_Float32 or GDT_CFloat32 depending on the type of calibration selected. The uncalibrated data is provided as GDT_Int16/GDT_Byte/GDT_CInt16, also depending on the type of product selected.

See Also:

- RadarSat document RN-RP-51-27.

ESRI ArcSDE Raster

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.

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.

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 disasterous) results.

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.

Terragen --- Terragen™ Terrain File

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.

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.

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`.

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.

History

v1.0 (Mar 26/06): Created.
v1.1 (Apr 20/06): Added Create() support and SIZE-only read fix.
v1.2 (Jun 6/07): Improved baseheight/scale determination when writing.

See Also:

- Implemented as `gdal/frmts/terragen/terragendataset.cpp`.
- See readme.txt for installation and support information.
- Terragen Terrain File Specification.

USGSDEM -- USGS ASCII DEM (and CDED)

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 georerenced 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.

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 govenment 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 (ie. 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.

GDAL Virtual Format Tutorial

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.

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 GDAL 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.

.vrt Format

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 SRS, GeoTransform, GCPList, Metadata, MaskBand and VRTRasterBand subelements.

```
<VRTDataset rasterXSize="512" rasterYSize="512">
```

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 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>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>
```

- **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>
  ```
- **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.

  ```
  <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. 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, 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>
```

- **Description**: This element contains the optional description of a raster band as it's 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. 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.

  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 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>
```

♦ **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
♦ **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.

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:

1. Nodata masking
2. Color table expansion
3. Applying the scale ratio
4. Applying the scale offset
5. 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>
```

♦ **KernelFilteredSource**: This 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. 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.

```
<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>
```

- ♦ **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.

.vrt Descriptions for Raw Files

So far we have described how to derive new virtual datasets from existing files supports 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 (ie. 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">
    <ColorInterp>Red</ColorInterp>
    <SourceFilename relativetoVRT="1">rgb.raw</SourceFilename>
    <ImageOffset>0</ImageOffset>
    <PixelOffset>3</PixelOffset>
    <LineOffset>1200</LineOffset>
  </VRTRasterBand>
  <VRTRasterBand dataType="Byte" band="2" subClass="VRTRawRasterBand">
    <ColorInterp>Green</ColorInterp>
    <SourceFilename relativetoVRT="1">rgb.raw</SourceFilename>
    <ImageOffset>1</ImageOffset>
    <PixelOffset>3</PixelOffset>
    <LineOffset>1200</LineOffset>
  </VRTRasterBand>
  <VRTRasterBand dataType="Byte" band="3" subClass="VRTRawRasterBand">
    <ColorInterp>Blue</ColorInterp>
    <SourceFilename relativetoVRT="1">rgb.raw</SourceFilename>
    <ImageOffset>2</ImageOffset>
    <PixelOffset>3</PixelOffset>
    <LineOffset>1200</LineOffset>
  </VRTRasterBand>
</VRTDataset>
```

Programatic 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" );

  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 );

      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");
// optionnal. default = 0
  papszOptions = CSLAddNameValue(papszOptions, "PixelOffset", "2");
// optionnal. default = size of band type
  papszOptions = CSLAddNameValue(papszOptions, "LineOffset", "1024");
// optionnal. default = size of band type * width
  papszOptions = CSLAddNameValue(papszOptions, "ByteOrder", "LSB");
// optionnal. default = machine order
  papszOptions = CSLAddNameValue(papszOptions, "RelativeToVRT", "true");
// optionnal. default = false
  poVRTDS->AddBand(GDT_Byte, papszOptions);
  CSLDestroy(papszOptions);

  delete poVRTDS;
```

Using Derived Bands

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>
    ...
```

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:

Parameters:

papoSources	A pointer to packed rasters; one per source. The datatype of all will be the same, specified in the eSrcType parameter.
nSources	The number of source rasters.
pData	The buffer into which the data should be read, or from which it should be written. This buffer must contain at least nBufXSize * nBufYSize words of type eBufType. It is organized in left to right, top to bottom pixel order. Spacing is controlled by the nPixelSpace, and nLineSpace parameters.
nBufXSize	The width of the buffer image into which the desired region is to be read, or from which it is to be written.
nBufYSize	The height of the buffer image into which the desired region is to be read, or from which it is to be written.
eSrcType	The type of the pixel values in the papoSources raster array.
eBufType	The type of the pixel values that the pixel function must generate in the pData data buffer.
nPixelSpace	The byte offset from the start of one pixel value in pData to the start of the next pixel value within a scanline. If defaulted (0) the size of the datatype eBufType is used.

nLineSpace	The byte offset from the start of one scanline in pData to the start of the next.

Returns:

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;
}
```

Multi-threading issues

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.

Various Support GDAL Raster Formats

AAIGrid -- Arc/Info ASCII Grid

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.

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 the first 100k bytes of data of the grid. From GDAL 1.8.0, you can explictely 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. For force the X pixel size to be used as CELLSIZE use the FORCE_CELLSIZE=YES creation option or resample the input to have square pixels.

When writing floating-point values, the driver uses the "%6.20g" 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.

The AIG driver is also available for Arc/Info Binary Grid format.

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

ACE2 -- ACE2

(GDAL >= 1.9.0)

This is a conveniency 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`.

ADRG/ARC Digitized Raster Graphics (.gen/.thf)

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).

This is an alternative to using the OGDI Bridge for ADRG datasets.

See also : the ADRG specification (MIL-A-89007)

AIG -- Arc/Info Binary Grid

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 format description 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`.

BSB -- Maptech/NOAA BSB Nautical Chart Format

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`.

BT -- VTP .bt Binary Terrain Format

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 writting 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.

CEOS -- CEOS Image

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`.

CTG -- USGS LULC Composite Theme Grid

(GDAL >= 1.9.0)

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`.

DIMAP -- Spot DIMAP

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`.

DODS/OPeNDAP -- Read rasters from DODS/OPeNDAP servers

Support for read access to DODS/OPeNDAP servers. Pass the DODS/OPeNDAP URL to the driver as you would when accessing a local file. The URL specifies the remote server, data set and raster within the data set. In addition, you must tell the driver which dimensions are to be interpreted as distinct bands as well as which correspond to Latitude and Longitude. See the file README.DODS for more detailed information.

DOQ1 -- First Generation USGS DOQ

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`.

DOQ2 -- New Labelled USGS DOQ

Support for read access, including reading of an affine georeferencing transform, capture of the projection string and reading of other auxilary 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

E00GRID -- Arc/Info Export E00 GRID

(GDAL >= 1.9.0)

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`.

EHdr -- ESRI .hdr Labelled

GDAL supports reading and writing the ESRI .hdr labelling 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. Unrecognised 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`.

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

ECRG Table Of Contents (TOC.xml)

Starting with GDAL 1.9.0

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.

Result of a gdalinfo on a TOC.xml file.

```
Subdatasets:
  SUBDATASET_1_NAME=ECRG_TOC_ENTRY:ECRG:FalconView:ECRG_Sample/EPF/TOC.xml
  SUBDATASET_1_DESC=ECRG:FalconView
```

See Also:

- NITF driver : format of the ECRG frames
- MIL-PRF-32283 : specification of ECRG products

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

EIR -- Erdas Imagine Raw

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`.

ENVI - ENVI .hdr Labelled Raster

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.

Creation Options:

- **INTERLEAVE=BSQ/BIP/BIL**: Force the generation specified type of interleaving. **BSQ** --- band sequental (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`.

Envisat -- Envisat Image Product

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.

FITS -- Flexible Image Transport System

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. Both reading and writing of FITS files is supported. At the current time, no support for a georeferencing system is implemented, but WCS (World Coordinate System) support is possible in the future.

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. Those header keywords are accessible and updatable via dataset metadata, in the same was as any other header keywords, but they do not affect reading/writing of data values from/to the file.

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

GenBin - Generic Binary (.hdr labelled)

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`.

GRASSASCIIGrid -- GRASS ASCII Grid

(GDAL >= 1.9.0)

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 explictely 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`.

GSAG -- Golden Software ASCII Grid File Format

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`.

GSBG -- Golden Software Binary Grid File Format

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`.

GS7BG -- Golden Software Surfer 7 Binary Grid File Format

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 format (also known as Surfer 6 binary grid format), it is more complicated and flexible. This format is supported for reading only.

NOTE: Implemented as `gdal/frmts/gsg/gs7bgdataset.cpp`.

GXF -- Grid eXchange File

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`.

IDA -- Image Display and Analysis

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 (ie. 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

ILWIS -- Raster Map

This driver implements reading and writing of ILWIS raster maps and map lists. Select the raster files with the.mpr (for raster map) or .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

GeoRefCorner 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.

JDEM -- Japanese DEM (.mem)

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.

LAN -- Erdas 7.x .LAN and .GIS

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 informaton. 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).

MFF -- Vexcel MFF Raster

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.

Unrecognised keywords from the .hdr file are preserved as metadata.

All data types with GDAL equivelents 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`.

NDF -- NLAPS Data Format

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.

GMT -- GMT Compatible netCDF

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

PAux -- PCI .aux Labelled Raw Format

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

PCRaster raster file format

GDAL includes support for reading and writing PCRaster raster files. PCRaster is a dynamic modelling system for distributed simulation models. The main applications of PCRaster are found in environmental modelling: geography, hydrology, ecology to name a few. Examples include rainfall-runoff models, vegetation competition models and slope stability 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 folowing 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 and PCRaster Environmental Software company website.

PNG -- Portable Network Graphics

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. World files 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 setted. 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.

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.

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.

PNM -- Netpbm (.pgm, .ppm)

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 plannig use the output files with software which is not liberal to this value.

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

Raster Product Format/RPF (a.toc)

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 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 wouldn't 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:

- OGDI Bridge : the RPFTOC driver gives an equivalent functionnality (without external dependency) to the RPF driver from the OGDI library.
- MIL-PRF-89038 : specification of RPF, CADRG, CIB products

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

SAR_CEOS -- CEOS SAR Image

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
  CEOS_ACQUISITION_TIME=19911029162818919
  CEOS_SENSOR_CLOCK_ANGLE=  90.000
  CEOS_ELLIPSOID=IUGG_75
  CEOS_SEMI_MAJOR=    6378.1400000
  CEOS_SEMI_MINOR=    6356.7550000
```

The SAR_CEOS driver also includes some support for SIR-C and PALSAR polarimetric data. The SIR-C format contains an image in compressed scattering matrix form, described here. GDAL decompresses the data as it is read in. The PALSAR format contains bands that correspond almost exactly to elements of the 3x3 Hermitian covariance matrix- see the ERSDAC-VX-CEOS-004A.pdf document for a complete description (pixel storage is described on page 193). GDAL converts these to complex floating point covariance matrix bands as they are read in. The convention used to represent the covariance matrix in terms of the scattering matrix elements HH, HV (=VH), and VV is indicated below. Note that the non-diagonal elements of the matrix are complex values, while the diagonal values are real (though represented as complex bands).

- Band 1: Covariance_11 (Float32) = HH*conj(HH)
- Band 2: Covariance_12 (CFloat32) = sqrt(2)*HH*conj(HV)
- Band 3: Covariance_13 (CFloat32) = HH*conj(VV)
- Band 4: Covariance_22 (Float32) = 2*HV*conj(HV)
- Band 5: Covariance_23 (CFloat32) = sqrt(2)*HV*conj(VV)
- Band 6: Covariance_33 (Float32) = VV*conj(VV)

The identities of the bands are also reflected in the metadata.

NOTE: Implemented as `gdal/frmts/ceos2/sar_ceosdataset.cpp`.

SDAT -- SAGA GIS Binary Grid File Format

(starting with GDAL 1.7.0)

The driver supports both reading and writing (including create, delete, and copy) SAGA GIS binary grids. SAGA binary grid datasets are made of an ASCII header (.SGRD) and a binary data (.SDAT) file with a common basename. The .SDAT file should be selected to access the dataset.

The driver 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`.

SDTS -- USGS SDTS DEM

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`.

SGI - SGI Image Format

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`.

SNODAS -- Snow Data Assimilation System

(GDAL >= 1.9.0)

This is a conveniency 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`.

Standard Product Format (ASRP/USRP) (.gen)

(starting with GDAL 1.7.0)

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.

NOTE: Implemented as `gdal/frmts/adrg/srpdataset.cpp`.

SRTMHGT - SRTM HGT Format

The SRTM HGT driver currently supports the reading of SRTM-3 and SRTM-1 V2 (HGT) files.

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`.

WLD -- ESRI World File

A world file file is a plain ASCII text file consisting of six values separated by newlines. The format is:

```
pixel X size
rotation about the Y axis (usually 0.0)
```

```
 rotation about the X axis (usually 0.0)
 negative pixel Y size
 X coordinate of upper left pixel center
 Y coordinate of upper left pixel center
```

For example:

```
60.0000000000
0.0000000000
0.0000000000
-60.0000000000
440750.0000000000
3751290.0000000000
```

You can construct that file simply by using your favorite text editor.

World file usually has suffix .wld, but sometimes it may has .tfw, tifw, .jgw or other suffixes depending on the image file it comes with.

XPM - X11 Pixmap

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 useable 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`.

GFF - Sandia National Laboratories GSAT File Format

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://sandia.gov/RADAR/sar-data.html.

The extension for GFF formats is .gff.

NOTE: Implemented as `gdal/frmts/gff/gff_dataset.cpp`.

ZMap -- ZMap Plus Grid

(GDAL >= 1.9.0)

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 interprete/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`.

Full list of GDAL Raster Formats

$Id: frmt_various.html 22861 2011-08-04 20:13:27Z rouault $

WCS -- OGC Web Coverage Service

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 xml file looking something like the following, with the coverage server url, and the name of the coverage to access. It is important that there be no spaces or other content before the `<WCS_GDAL>` element.

```
<WCS_GDAL>
  <ServiceURL>http://laits.gmu.edu/cgi-bin/NWGISS/NWGISS?</ServiceURL>
  <CoverageName>AUTUMN.hdf</CoverageName>
</WCS_GDAL>
```

When first opened, GDAL will fetch the coverage description, and a small test image to establish details about the raster. This information will be cached in the service description file to make future opens faster - no server access should be required till imagery is read for future opens.

The WCS driver should support WCS 1.0.0 and 1.1.0 servers, but WCS 0.7 servers are not supported. 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. Coordinate systems are read from the DescribeCoverage result, and are expected to be in the form of `EPSG:n` in the `<supportedCRSs>` element.

The service description file has the following additional elements as immediate children of the `WCS_GDAL` element that may be optionally set.

- **PreferredFormat**: the format to use for GetCoverage calls.
- **BandCount**: Number of bands in the dataset, normally captured from the sample request.
- **BandType**: The pixel data type to use. Normally established from the sample request.
- **BlockXSize**: The block width to use for block cached remote access.
- **BlockYSize**: The block height to use for block cached remote access.
- **NoDataValue**: The nodata value to use for all the bands (blank for none). Normally defaulted from CoverageOffering info.
- **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.
- **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).
- **GetCoverageExtra**: An additional set of keywords to add to GetCoverage requests in URL encoded form. eg. "&RESAMPLE=BILINEAR&Band=1"
- **DescribeCoverageExtra**: An additional set of keywords to add to DescribeCoverage requests in URL encoded form. eg. "&CustNo=775"
- **Version**: Set a specific WCS version to use. Currently defaults to 1.0.0 and 1.1.0 is also supported.
- **FieldName**: Name of the field being accessed. Used only with WCS 1.1.0+. Defaults to the first field in the DescribeCoverage result.
- **DefaultTime**: A timePosition to use by default when accessing coverages with a time dimension. Populated with the last offered time position by default.

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.

See Also:

- OGC WCS Standards

WEBP - WEBP

Starting with GDAL 1.9.0, 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 licenced). 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.

The WEBP driver only supports 3 bands (RGB) images

The WEBP driver can be used as the internal format used by the Rasterlite driver.

Creation options

- **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.

See Also:

- WebP home page

WMS -- Web Map Services

Accessing several different types of web image services is possible using the WMS format in GDAL. Services are accessed by creating a local service description XML file -- there are examples below for each of the supported image services. It is important that there be no spaces or other content before the `<GDAL_WMS>` element.

<GDAL_WMS>	
<Service name="WMS">	Define what mini-driver to use, currently supported are: WMS, WorldWind, TileService, TMS, TiledWMS or VirtualEarth. (required)
<Version>1.1.1</Version>	WMS version. (optional, defaults to 1.1.1)
<ServerUrl>http://onearth.jpl.nasa.gov/wms.cgi?</ServerUrl>	WMS server URL. (required)
<SRS>EPSG:4326</SRS>	Image projection (optional, defaults to EPSG:4326, WMS version 1.1.1 or below only)
<CRS>CRS:83</CRS>	Image projection (optional, defaults to EPSG:4326, WMS version 1.3.0 or above only)
<ImageFormat>image/jpeg</ImageFormat>	Format in which to request data. Paletted formats like image/gif will be converted to RGB. (optional, defaults to image/jpeg)
<Transparent>FALSE</Transparent>	Set to TRUE to include "transparent=TRUE" in the WMS GetMap request (optional defaults to FALSE). The request format and BandsCount need to support alpha.
<Layers>modis,global_mosaic</Layers>	Comma separated list of layers. (required, except for TiledWMS)
<TiledGroupName>Clementine</TiledGroupName>	Comma separated list of layers. (required for TiledWMS)
<Styles></Styles>	Comma separated list of styles. (optional)
<BBoxOrder>xyXY</BBoxOrder>	Reorder bbox coordinates arbitrarly. May be required for version 1.3 servers. (optional) x - low X coordinate, y - low Y coordinate, X - high X coordinate, Y - high Y coordinate
</Service>	
<DataWindow>	Define size and extents of the data. (required, except for TiledWMS and VirtualEarth)
<UpperLeftX>-180.0</UpperLeftX>	X (longitude) coordinate of upper-left corner. (optional, defaults to -180.0, except for VirtualEarth)
<UpperLeftY>90.0</UpperLeftY>	Y (latitude) coordinate of upper-left corner. (optional, defaults to 90.0, except for VirtualEarth)
<LowerRightX>180.0</LowerRightX>	X (longitude) coordinate of lower-right corner. (optional, defaults to 180.0, except for VirtualEarth)
<LowerRightY>-90.0</LowerRightY>	Y (latitude) coordinate of lower-right corner. (optional, defaults to -90.0, except for VirtualEarth)
<SizeX>2666666</SizeX>	Image size in pixels.
<SizeY>1333333</SizeY>	Image size in pixels.
<TileX>0</TileX>	

	Added to tile X value at highest resolution. (ignored for WMS, tiled image sources only, optional, defaults to 0)
<TileY>0</TileY>	Added to tile Y value at highest resolution. (ignored for WMS, tiled image sources only, optional, defaults to 0)
<TileLevel>0</TileLevel>	Tile level at highest resolution. (tiled image sources only, optional, defaults to 0)
<TileCountX>0</TileCountX>	Can be used to define image size, SizeX = TileCountX * BlockSizeX * $2^{TileLevel}$. (tiled image sources only, optional, defaults to 0)
<TileCountY>0</TileCountY>	Can be used to define image size, SizeY = TileCountY * BlockSizeY * $2^{TileLevel}$. (tiled image sources only, optional, defaults to 0)
<YOrigin>top</YOrigin>	Can be used to define the position of the Y origin with respect to the tile grid. Possible values are 'top', 'bottom', and 'default', where the default behavior is mini-driver-specific. (TMS mini-driver only, optional, defaults to 'bottom' for TMS)
</DataWindow>	
<Projection>EPSG:4326</Projection>	Image projection (optional, defaults to value reported by mini-driver or EPSG:4326)
<BandsCount>3</BandsCount>	Number of bands/channels, 1 for grayscale data, 3 for RGB. (optional, defaults to 3)
<BlockSizeX>1024</BlockSizeX>	Block size in pixels. (optional, defaults to 1024, except for VirtualEarth)
<BlockSizeY>1024</BlockSizeY>	Block size in pixels. (optional, defaults to 1024, except for VirtualEarth)
<OverviewCount>10</OverviewCount>	Count of reduced resolution layers each having 2 times lower resolution. (optional, default is calculated at runtime)
<Cache>	Enable local disk cache. Allows for offline operation. (optional, defaults to no cache)
<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)
<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. Usefull only with cache enabled. (optional, defaults to false)

<AdviseRead>true</AdviseRead>	Enable AdviseRead API call - download images into cache. (optional, defaults to false)
<VerifyAdviseRead>true</VerifyAdviseRead>	Open each downloaded image and do some basic checks before writing into cache. Disabling can save some CPU cycles if server is trusted to always return correct images. (optional, defaults to true)
<ClampRequests>false</ClampRequests>	Should requests, that otherwise would be partially outside of defined data window, be clipped resulting in smaller than block size request. (optional, defaults to true)
<UserAgent>GDAL WMS driver (http://www.gdal.org/frmt_wms.html)</UserAgent>	HTTP User-agent string. Some servers might require a well-known user-agent such as "Mozilla/5.0" (optional, defaults to "GDAL WMS driver (http://www.gdal.org/frmt_wms.html)"). Added in GDAL 1.8.0
<Referer>http://example.foo/</Referer>	HTTP Referer string. Some servers might require it (optional). Added in GDAL 1.9.0
</GDAL_WMS>	

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.

WMS

Communications with an OGC WMS server. Has support for both tiled and untiled requests.

TileService

Service to support talking to a WorldWind TileService. Access is always tile based.

WorldWind

Access to web-based WorldWind tile services. Access is always tile based.

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://labs.metacarta.com/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.

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.

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

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 one
  ```
- 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
  ```
- tileservice_bmng.xml - TileService, Blue Marble NG (January)

  ```
  gdal_translate -of JPEG -outsize 500 250 tileservice_bmng.xml tileservice_bmng.jpg
  ```
- tileservice_nysdop2004.xml - TileService, NYSDOP 2004

  ```
  gdal_translate -of JPEG -projwin -73.687030 41.262680 -73.686359 41.262345 -outsize 500 250
  ```
- 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. `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.
- Swiss Geoportal maps accessed with the TMS minidriver (needs GDAL >= 1.9.0)
- OnEarth Tiled WMS Clementine, daily, and srtm examples.
- VirtualEarth Aerial Layer accessed with the VirtualEarth minidriver.

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=WMSVERSION=
  1.1.1REQUEST=GetMapLAYERS=DNEC_250K:ELEVATION/ELEVATIONSRS=EPSG:42304BBOX=-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 >= 1.9.0) the URL of a REST definition for a ArcGIS MapServer:

```
gdalinfo "http://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/
    MapServer?f=jsonpretty=true"
```

See Also:

- OGC WMS Standards
- WMS Tiling Client Recommendation (WMS-C)
- WorldWind TileService
- TMS Specification
- OnEarth Tiled WMS specification

XYZ -- ASCII Gridded XYZ

(Available for GDAL >= 1.8.0)

GDAL supports reading and writting ASCII **gridded** XYZ raster datasets (ie. 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.

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

See also

- Documentation of gdal_grid utility

OGR Utility Programs

The following utilities are distributed as part of the OGR Simple Features toolkit:

- ogrinfo - Lists information about an OGR supported data source
- ogr2ogr - Converts simple features data between file formats
- ogrtindex - Creates a tileindex

ogrinfo

lists information about an OGR supported data source

Usage:

```
ogrinfo [--help-general] [-ro] [-q] [-where restricted_where]
        [-spat xmin ymin xmax ymax] [-fid fid]
        [-sql statement] [-dialect dialect] [-al] [-so] [-fields={YES/NO}]
        [-geom={YES/NO/SUMMARY}][--formats]
        datasource_name [layer [layer ...]]
```

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).

-so:
: Summary Only: supress listing of features, show only the 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.

-sql *statement*:
: Execute the indicated SQL statement and return the result.

-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.

-spat *xmin ymin xmax ymax*:
: The area of interest. Only features within the rectangle will be reported.

-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}:
: (starting with GDAL 1.6.0) If set to NO, the feature dump will not display field values. Default value is YES.

-geom={YES/NO/SUMMARY}:
: (starting with GDAL 1.6.0) 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, the geometry will be reported in full OGC WKT format. Default value is YES.

--formats:
: List the format drivers that are enabled.

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 layerwide 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.

Example reporting all layers in an NTF file:

```
% ogrinfo wrk/SHETLAND_ISLANDS.NTF
INFO: Open of `wrk/SHETLAND_ISLANDS.NTF'
using driver `UK .NTF' successful.
1: BL2000_LINK (Line String)
2: BL2000_POLY (None)
3: BL2000_COLLECTIONS (None)
4: FEATURE_CLASSES (None)
```

Example 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)
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)
```

ogr2ogr

converts simple features data between file formats

Usage:

```
Usage: ogr2ogr [--help-general] [-skipfailures] [-append] [-update]
               [-select field_list] [-where restricted_where]
               [-progress] [-sql <sql statement>] [-dialect dialect]
               [-preserve_fid] [-fid FID]
               [-spat xmin ymin xmax ymax]
               [-a_srs srs_def] [-t_srs srs_def] [-s_srs srs_def]
               [-f format_name] [-overwrite] [[-dsco NAME=VALUE] ...]
               dst_datasource_name src_datasource_name
               [-lco NAME=VALUE] [-nln name] [-nlt type] [layer [layer ...]]

Advanced options :
               [-gt n]
               [-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]
               [-clipdstwhere expression]
               [-wrapdateline]
               [-segmentize max_dist] [-fieldTypeToString All|(type1[,type2]*)]
               [-splitlistfields] [-maxsubfields val]
               [-explodecollections] [-zfield field_name]
```

This program can be used to convert simple features data between file formats performing various operations during the process such as spatial or attribute selections, reducing the set of attributes, setting the output coordinate system or even reprojecting the features during translation.

-f *format_name*:
: output file format name (default is ESRI Shapefile), some possible values are:

```
     -f "ESRI Shapefile"
     -f "TIGER"
     -f "MapInfo File"
     -f "GML"
     -f "PostgreSQL"
```

-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.)

-progress:
: (starting with GDAL 1.7.0) 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.

-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.

-where *restricted_where*:
: Attribute query (like SQL WHERE)

-skipfailures:
: Continue after a failure, skipping the failed feature.

-spat *xmin ymin xmax ymax*:
: spatial query extents. Only features whose geometry intersects the extents will be selected. The geometries will not be clipped unless -clipsrc is specified

-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 or MULTILINESTRING. Add "25D" to the name to get 2.5D versions.

-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

-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.

Srs_def can be a full WKT definition (hard to escape properly), or a well known definition (ie. EPSG:4326) or a file with a WKT definition.

Advanced options :

-gt *n*:
: group *n* features per transaction (default 200)

-clipsrc *[xmin ymin xmax ymax]|WKT|datasource|spat_extent*:
: (starting with GDAL 1.7.0) 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 **-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.

-clipdst *xmin ymin xmax ymax*:
: (starting with GDAL 1.7.0) 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:
: (starting with GDAL 1.7.0) split geometries crossing the dateline meridian (long. = +/- 180deg)

-segmentize *max_dist*:
: (starting with GDAL 1.6.0) maximum distance between 2 nodes. Used to create intermediate pointsspatial query extents

-fieldTypeToString *type1, ...*:
: (starting with GDAL 1.7.0) converts any field of the specified type to a field of type string in the destination layer. Valid types are : Integer, Real, String, Date, Time, DateTime, Binary, IntegerList, 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.

-splitlistfields:
: (starting with GDAL 1.8.0) 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:
: (starting with GDAL 1.8.0) produce one feature for each geometry in any kind of geometry collection in the source file

-zfield *field_name*:
: (starting with GDAL 1.8.0) Uses the specified field to fill the Z coordinate of geometries

Example appending to an existing layer (both flags need to be used):

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

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
```

More examples are given in the individual format pages.

ogrtindex

creates a tileindex

Usage:

```
ogrtindex [-lnum n]... [-lname name]... [-f output_format]
          [-write_absolute_path] [-skip_different_projection]
                 output_dataset src_dataset...
```

The ogrtindex program can be used to create a tileindex - a file containing a list of the identities of a bunch of other files along with there 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.

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.

It is a flaw of the current ogrtindex program that no attempt is made to copy the coordinate system definition from the source datasets to the tile index (as is expected by MapServer when PROJECTION AUTO is in use).

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
```

OGR Vector Formats

Format Name	Code	Creation	Georeferencing	Compiled by default
Aeronav FAA files	AeronavFAA	No	Yes	Yes
ESRI ArcObjects	ArcObjects	No	Yes	No, needs ESRI ArcObjects
Arc/Info Binary Coverage	AVCBin	No	Yes	Yes
Arc/Info .E00 (ASCII) Coverage	AVCE00	No	Yes	Yes
Atlas BNA	BNA	Yes	No	Yes
AutoCAD DXF	DXF	Yes	No	Yes
Comma Separated Value (.csv)	CSV	Yes	No	Yes
CouchDB / GeoCouch	CouchDB	Yes	Yes	No, needs libcurl
DODS/OPeNDAP	DODS	No	Yes	No, needs libdap
EDIGEO	EDIGEO	No	Yes	Yes
ESRI FileGDB	FileGDB	Yes	Yes	No, needs FileGDB API library
ESRI Personal GeoDatabase	PGeo	No	Yes	No, needs ODBC library
ESRI ArcSDE	SDE	No	Yes	No, needs ESRI SDE
ESRI Shapefile	ESRI Shapefile	Yes	Yes	Yes
FMEObjects Gateway	FMEObjects Gateway	No	Yes	No, needs FME
GeoJSON	GeoJSON	Yes	Yes	Yes
Geoconcept Export	Geoconcept	Yes	Yes	Yes
Geomedia .mdb	Geomedia	No	No	No, needs ODBC library
GeoRSS	GeoRSS	Yes	Yes	Yes (read support needs libexpat)
Google Fusion Tables	GFT	Yes	Yes	No, needs libcurl
GML	GML	Yes	Yes	Yes (read support needs Xerces or libexpat)
GMT	GMT	Yes	Yes	Yes
GPSBabel	GPSBabel	Yes	Yes	Yes (needs GPSBabel and GPX driver)
GPX	GPX	Yes	Yes	Yes (read support needs libexpat)
GRASS	GRASS	No	Yes	No, needs libgrass
GPSTrackMaker (.gtm, .gtz)	GPSTrackMaker	Yes	Yes	Yes
Hydrographic Transfer Format	HTF	No	Yes	Yes
Informix DataBlade	IDB	Yes	Yes	No, needs Informix DataBlade
INTERLIS	"Interlis 1" and "Interlis 2"	Yes	Yes	No, needs Xerces (INTERLIS model reading needs ili2c.jar)
INGRES	INGRES	Yes	No	No, needs INGRESS
KML	KML	Yes	Yes	Yes (read support needs libexpat)
LIBKML	LIBKML	Yes	Yes	No, needs libkml
MapInfo File	MapInfo File	Yes	Yes	Yes

Microstation DGN	DGN	Yes	No	Yes
Access MDB (PGeo and Geomedia capable)	MDB	No	Yes	No, needs JDK/JRE
Memory	Memory	Yes	Yes	Yes
MySQL	MySQL	No	Yes	No, needs MySQL library
NAS - ALKIS	NAS	No	Yes	No, needs Xerces
Oracle Spatial	OCI	Yes	Yes	No, needs OCI library
ODBC	ODBC	No	Yes	No, needs ODBC library
MS SQL Spatial	MSSQLSpatial	Yes	Yes	No, needs ODBC library
OGDI Vectors (VPF, VMAP, DCW)	OGDI	No	Yes	No, needs OGDI library
OpenAir	OpenAir	No	Yes	Yes
PCI Geomatics Database File	PCIDSK	No	No	Yes, using internal PCIDSK SDK (from GDAL 1.7.0)
PDS	PDS	No	Yes	Yes
PGDump	PostgreSQL SQL dump	Yes	Yes	Yes
PostgreSQL/PostGIS	PostgreSQL/PostGIS	Yes	Yes	No, needs PostgreSQL client library (libpq)
EPIInfo .REC	REC	No	No	Yes
S-57 (ENC)	S57	No	Yes	Yes
SDTS	SDTS	No	Yes	Yes
Norwegian SOSI Standard	SOSI	No	Yes	No, needs FYBA library
SQLite/SpatiaLite	SQLite	Yes	Yes	No, needs libsqlite3 or libspatialite
SUA	SUA	No	Yes	Yes
SVG	SVG	No	Yes	No, needs libexpat
UK .NTF	UK. NTF	No	Yes	Yes
U.S. Census TIGER/Line	TIGER	No	Yes	Yes
VFK data	VFK	No	Yes	Yes
VRT - Virtual Datasource	VRT	No	Yes	Yes
OGC WFS (Web Feature Service)	WFS	Yes	Yes	No, needs libcurl
X-Plane/Flighgear aeronautical data	XPLANE	No	Yes	Yes

Aeronav FAA

(GDAL/OGR >= 1.8.0)

This driver reads text files describing aeronav information - obstacles, navaids and routes - as provided by the FAA.

See Also

- Digital Obstacle File
- NAVAID Digital Data File
- Digital Aeronautical Chart Supplement

ESRI ArcObjects

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.

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

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"
```

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
```

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.

Arc/Info Binary Coverage

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 sepecifically named to be attached to one of the existing geometric layers is currently not accessable through OGR. Note that info tables are stored in an 'info' directory at the same level as the coverage directory. If this is inaccessable or corrupt no info attributes will be appended to coverage layers, but the geometry should still be accessable.

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.

See Also

- AVCE00 Library Page
- AVCE00 OGR Driver (.E00)

Arc/Info E00 (ASCII) Coverage

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 sepecifically named to be attached to one of the existing geometric layers is currently not accessable through OGR. Note that info tables are stored in an 'info' directory at the same level as the coverage directory. If this is inaccessable or corrupt no info attributes will be appended to coverage layers, but the geometry should still be accessable.

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 ascess to E00 files is generally slow.

See Also

- AVCE00 Library Page
- AVCBin OGR Driver (Binary Coverage)

BNA - Atlas BNA

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 it 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 support 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.

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 exits, 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 contains 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.

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
```

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

CouchDB - CouchDB/GeoCouch

(GDAL/OGR >= 1.9.0)

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.

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.

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, optionaly, 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.

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.

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.

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.

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 successfull, OGR will fetch the returned _id and _rev and use them.

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.

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 writting 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.

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
```

See Also

- CouchDB reference
- GeoCouch source code repository
- Documentation for 'validate_doc_update' function

Comma Separated Value (.csv)

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 extention ".csv". The datasource name may be either a single CSV file or 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 never have any coordinate system. 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 with a type of string 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) and DateTime (YYYY-MM-DD HH:MM:SS+nn) columns through a descriptive file with same name as the CSV file, but .csvt extension. In a single line the types for each column have to be listed: double ed and comma separated (e.g., "Integer","String"). It is also possible to specify explicitely 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.

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. Starting with GDAL 1.7.0, the driver will also accept a semicolon or a tabulation 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, es or newlines) may be placed in double es. Any occurances of double es within the ed string should be doubled up to "escape" them.

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.

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 es because the value contains es, and those es have to be doubled up so we know we haven't reached the end of the ed 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 seperators or those with other auxilary 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.

Reading CSV containing spatial information

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

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"
```

You can write the associated VRT file (test.vrt):

```
<OGRVRTDataSource>
    <OGRVRTLayer name="test">
        <SrcDataSource>test.csv</SrcDataSource>
        <GeometryType>wkbPoint</GeometryType>
        <LayerSRS>WGS84</LayerSRS>
        <GeometryField encoding="PointFromColumns" x="Longitude" y="Latitude"/>
    </OGRVRTLayer>
</OGRVRTDataSource>
```

and *ogrinfo -ro -al test.vrt* will return :

```
OGRFeature(test):1
  Latitude (String) = 48.1
  Longitude (String) = 0.25
  Name (String) = First point
  POINT (0.25 48.1 0)

OGRFeature(test):2
  Latitude (String) = 49.2
  Longitude (String) = 1.1
  Name (String) = Second point
  POINT (1.1 49.200000000000003 0)

OGRFeature(test):3
  Latitude (String) = 47.5
  Longitude (String) = 0.75
  Name (String) = Third point
  POINT (0.75 47.5 0)
```

Creation Issues

The driver supports creating new databases (as a directory of .csv files), adding new .csv files to an existing directory or .csv files or appending features to an existing .csv table. Deleting or replacing existing features is not supported.

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.

- **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 (Starting with GDAL 1.7.0): Field separator character. Default value : COMMA

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
```

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 tha 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)

Other Notes

- Development of the OGR CSV driver was supported by DM Solutions Group and GoMOOS.

Microstation DGN

Microstation DGN files from Microstation versions predating version 8.0 are supported for reading. 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.

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.

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 geomtries will carry style information indicating the color, thickness and line style (ie. 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.

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 elmements 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_TABLEE=***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
- OGR Feature Style Specification

DODS/OPeNDAP

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, it's name should be specified in the form "DODS:url". The URL may include a constraint expression a shown here. Note that it may be necessary to e 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 auxilary informaton) 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;
        }
    }
}
```

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.

See Also

- OPeNDAP

AutoCAD DXF

OGR supports reading most versions of AutoCAD DXF and writing AutoCAD 2000 version files. DXF is an ASCII format used for interchanging AutoCAD drawings between different software packages. The entire contents of the file is represented as a single layer named "entities".

DXF 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.

Supported Elements

The following element types are supported:

- POINT: Produces a simple point geometry feature.
- MTEXT, TEXT: Produces a point feature with LABEL style information.
- LINE, POLYLINE, LWPOLYLINE: translated as a LINESTRING or POLYGON depending on whether it is closed. Rounded polylines (those with their vertices' budge attributes set) will be tesselated. Single-vertex polylines are translated to POINT.
- CIRCLE, ELLIPSE, ARC: Translated as a LINESTRING, tesselating the arc into line segments.
- INSERT: An attempt is made to insert the block definition as defined in the insert. Linework blocks are aggregated into a single feature with a geometry collection for the geometry. Text blocks are returned as one or more text features. To avoid merging blocks into a geometry collection the DXF_MERGE_BLOCK_GEOMETRIES config option may be set to FALSE.
- DIMENSION: This element is exploded into a feature with arrows and leaders, and a feature with the dimension label.
- 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.

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 threshhold 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.

DXF_INLINE_BLOCKS

The default behavior is for INSERT entities to be expanded 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 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.

- 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.

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() (ie. 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.

Creation Issues

DXF files are written in AutoCAD 2000 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 layer can be created on the output file.

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, MultiLineString, Polygon and MultiPolygons are written as one or more LWPOLYLINE entities, closed in the case of polygon rings. An effort is made to preserve line width and color.

The dataset creation supports the following dataset creation options:

- **HEADER=***filename*: Override the header file used - in place of header.dxf.
- **TRAILER=***filename*: Override the trailer file used - in place of trailer.dxf.

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).

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 follow 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 BlockName field populated.
- Objects to be written as INSERTs in the entities layer should have a POINT geometry, and the BlockName field set.

- 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 with DXF_INLINE_BLOCKS set to FALSE will approximately reproduce the original blocks and keep INSERT entities as INSERT entities rather than exploding them.

Layer Definitions

When writing entities, if populated the LayerName 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").

Line Type Definitions When writing LWPOLYLINE entities the following rules apply with regard to Linetype definitions.

- If the Linetype field is set on the written features, and that Linetype is already defined in the template header then it will be referenced from the entities regardless of whether an OGR style string existed.
- If the Linetype is set, but the Linetype is not predefined 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.
- 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.

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.

EDIGEO

(GDAL/OGR >= 1.9.0)

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 modelling, etc.

The driver has been developed to read files of the French PCI (Plan Cadastral Informatisé - Digital Cadastral Plan) as produced by the DGI (Direction Générale des Impots - General Tax Office). The driver should also be able to open other EDIGEO based products.

The driver must be provided with the .THF file describing the EDIGEO exchange and it will read the associated .DIC, .GEO, .SCD, .QAL and .VEC files.

In order the SRS of the layers to be correctly built, the IGNF file that contains the defintion of IGN SRS must be placed in the directory of PROJ.4 ressource files.

The whole set of files will be parsed into memory. This may be a limitation if dealing with big EDIGEO exchanges.

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 OGR Feature Style specification.

It will also add the following fields :

- OGR_OBJ_LNK: the id of the object that is linked to this label
- OBJ_OBJ_LNK_LAYER: the name of the layer of the linked object
- OGR_ATR_VAL: the value of the attribute to display (found in the ATR attribute of the OGR_OBJ_LNK object)
- OGR_ANGLE: the rotation angle in degrees (0 = horizontal, counter-clock-wise oriented)
- OGR_FONT_SIZE: the value of the HEI attribute multiplied by the value of the configuration option OGR_EDIGEO_FONT_SIZE_FACTOR 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.

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

ESRI File Geodatabase (FileGDB)

The FileGDB driver provides read and write access to File Geodatabases (.gdb directories) created by ArcGIS 10 and above.

Requirements

- FileGDB API SDK
- OGR >= 1.9.0

Dataset Creation Options

None.

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.
- **GEOMETRY_NAME**: Set name of geometry column in new layer. Defaults to "SHAPE".
- **OID_NAME**: Name of the OID column to create. Defaults to "OBJECTID".
- **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.
- **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 by used by ArcGIS. The units of the parameters are the units of the coordinate reference system.

Examples

- Read layer from FileGDB and load into PostGIS:

  ```
  ogr2ogr -overwrite -skipfailures -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"
  ```

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\ogrsf_frmts\filegdb* folder and execute:

```
nmake /f makefile.vc plugin nmake /f makefile.vc plugin-install
```

Known Issues

- Blob fields have not been implemented.
- FGDB coordinate snapping will cause geometries to be altered during writing. Use the origin and scale layer creation options to control the snapping behavior.

Links

- ESRI File Geodatabase API Page

FMEObjects Gateway

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 accessable.

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 (ie. the filename). For instance, "NTF:F:\DATA\NTF\2144.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.

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 featur 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 rentention 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.

Caveats

1. Establishing an FME session is quite an expensive operation, on a 350Mhz Linux system this can be in exceeess 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 indefinately.

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.

GeoConcept text export

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.

GeoConcept Text File Format (gxt)

GeoConcept is a GIS developped 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.

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.

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 og 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.

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 decimales for cartesian spatial reference systems (including height) or with 9 decimales for geographical spatial reference systems.

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
//# 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 :

```
//$DELIMITER &; &;
//$ED-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
-1      TILE    TILE    TILE    3       000-2007-0050-7130-LAMB93 0  50000.00
-1      TILE    TILE    TILE    3       000-2007-0595-7130-LAMB93 0  595000.00
-1      TILE    TILE    TILE    3       000-2007-0595-6585-LAMB93 0  595000.00
1       TILE    TILE    TILE    3       000-2007-1145-6250-LAMB93 0  1145000.00
1       TILE    TILE    TILE    3       000-2007-0050-6585-LAMB93 0  50000.00

Private#Name Private#NbFields IDSEL  NOM  WITHDATA  Private#X  Private#Y        Private#Graphics
4  600000.00  7130000.00  600000.00  6580000.00  50000.00  6580000.00 50000.00  7130000.00
4  1145000.00  7130000.00  1145000.00    6580000.00  595000.00  6580000.00  595000.00  7130000.00
4  1145000.00  6585000.00  1145000.00    6035000.00  595000.00 6035000.00 595000.00  6585000.00
4  1265000.00  6250000.00  1265000.00    6030000.00  1145000.006030000.00  1145000.00  6250000.00
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
```

See Also

- GeoConcept web site

GeoJSON

This driver implements read/write support for access to features encoded in GeoJSON format. The GeoJSON is a dialect based on the JavaScript Object Notation (JSON). The JSON is a lightweight plain text format for data interchange and GeoJSON is nothing other than its specialization for geographic content.

At the moment of writing this, GeoJSON is supported as output format of services implemented by FeatureServer, GeoServer and CartoWeb.

The OGR GeoJSON driver translates a GeoJSON encoded data to objects of OGR Simple Features model: Datasource, Layer, Feature, Geometry. The implementation is based on GeoJSON Specification draft, v5.0.

Starting with OGR 1.8.0, the GeoJSON driver can read the JSON output of Feature Service request following the GeoServices REST Specification, like implemented by ArcGIS Server REST API

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

Layer

A GeoJSON datasource is translated to single OGRLayer object with pre-defined name *OGRGeoJSON*:

```
ogrinfo -ro http://featureserver/data/.geojson OGRGeoJSON
```

It's also valid to assume that OGRDataSource::GetLayerCount() for GeoJSON datasource always returns 1.

Accessing Web Service as a datasource (ie. 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.

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 must 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.

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**.

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 homogeneous layer from a set of heterogeneous features. For this purpose, it's possible to tell the driver to wrap all geometries with OGRGeometryCollection type as a common denominator. This behavior may be controlled by setting environment variable **GEOMETRY_AS_COLLECTION=YES** (default is **NO**).

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

Layer creation option

- **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. "Smart" truncation will occur to remove trailing zeros.

Example

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
```

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+objectidoutfields=*f=json"
```

See Also

- GeoJSON - encoding geographic content in JSON
- JSON - JavaScript Object Notation
- JSON-C - A JSON implementation in C
- [Gdal-dev] OGR GeoJSON Driver - driver announcement
- GeoServices REST Specification

Geomedia MDB database

GDAL/OGR >= 1.9.0

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 drivers 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.

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:

See also

- MDB driver page

GeoRSS : Geographically Encoded Objects for RSS feeds

(Driver available in GDAL 1.7.0 or later)

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 compatibily 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.

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 mentionned 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 mentionned 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.

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 usefull 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 descriprion !</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 descriprion !
  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)
```

Creation Issues

On export, all layers are written to a single file. Update of existing files is not supported.

If the output file already exits, the writing will not occur. You have to delete the existing file first.

A layer that is created cannot be immediately read without closing and reopening the file. That is to say that a dataset is read-only or write-only in the same session.

Supported geometries :

- Features of type wkbPoint/wkbPoint25D.

- Features of type wkbLineString/wkbLineString25D.
- Features of type wkbPolygon/wkbPolygon25D.

Other type of geometries are not supported and will be silently ignored.

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 RFC21: OGR SQL type cast and field name alias)

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,
  ```

  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)
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')
```

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

GFT - Google Fusion Tables

(GDAL/OGR >= 1.9.0)

This driver can connect to the Google Fusion Tables service. GDAL/OGR must be built with Curl support in order to the GFT driver to be compiled.

The driver supports read and write operations.

Dataset name syntax

The minimal syntax to open a GFT datasource is :

```
GFT:
```

Additionnal optional parameters can be specified after the ':' sign such as :

- **tables=table_id1[,table_id2]***: A list of table IDs. This is necessary when you need to access to public tables for example. If you want the table ID of a public table, or any other table that is not owned by the authenticated user, you have to visit the table in the Google Fusion Tables website and note the number at the end of the URL.
- **protocol=https**: To use HTTPS protocol for all operations. By default, HTTP is used, except for the authentication operation where HTTPS is always used.
- **email=john.doe@example.com**: The email of the Google account used for authentication.
- **password=your_secret_password**: The password of the Google account used for authentication.
- **auth=auth_key**: An authentication key as described below.

If several parameters are specified, they must be separated by a space.

Authentication

Most operations, in particular write operations, require a valid Google account to provide authentication information to the driver. The only exception is read-only access to public tables.

The authentication information can be provided by different means :

- specifying the *email* and *password* parameters in the dataset name, as described above.
- specifying them through the *GFT_EMAIL* and *GFT_PASSWORD* configuration options/environment variables.
- specifying the authentication key through the *GFT_AUTH* configuration option/environment variable. This value is a key returned by the server when using the email + password method, and that can re-used for later authentication. You can fetch this key by running a first time ogrinfo with CPL_DEBUG set to ON and authenticating with another method. The value will be displayed after the "Auth key : " string. This method has the advantage of not exposing your plain password.
- specifying the authentication key through the *auth* connection parameter.

Geometry

Geometries in GFT tables must be expressed in the geodetic WGS84 SRS. GFT allows them to be encoded in different forms :

- A single column with a "lat lon" or "lat,lon" string, where lat et lon are expressed in decimal degrees.
- A single column with a KML string that is the representation of a Point, a LineString or a Polygon.

- Two columns, one with the latitude and the other one with the longitude, both values expressed in decimal degrees.
- A single column with an address known by the geocoding service of Google Maps.

Only the first 3 types are supported by OGR, not the last one.

Fusion tables can have multiple geometry columns per table. By default, OGR will use the first geometry column it finds. It is possible to select another column as the geometry column by specifying *table_name(geometry_column_name)* as the layer name passed to GetLayerByName().

Filtering

The driver will forward any spatial filter set with SetSpatialFilter() to the server. It also makes the same for attribute filters set with SetAttributeFilter().

Paging

Features are retrieved from the server by chunks of 500 by default. This number can be altered with the GFT_PAGE_SIZE configuration option.

Write support

Table creation and deletion is possible. Note that fields can only be added to a table in which there are no features created yet.

Write support is only enabled when the datasource is opened in update mode.

The mapping between the operations of the GFT service and the OGR concepts is the following :

- OGRFeature::CreateFeature() <==> INSERT operation
- OGRFeature::SetFeature() <==> UPDATE operation
- OGRFeature::DeleteFeature() <==> DELETE operation
- OGRDataSource::CreateLayer() <==> CREATE TABLE operation
- OGRDataSource::DeleteLayer() <==> DROP TABLE operation

When inserting a new feature with CreateFeature(), and if the command is successfull, OGR will fetch the returned rowid and use it as the OGR FID. OGR will also automatically reproject its geometry into the geodetic WGS84 SRS if needed (provided that the original SRS is attached to the geometry).

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 the CreateFeature() oepration between OGRLayer::StartTransaction() and OGRLayer::CommitTransaction(). The operations will be stored into memory and only executed at the time CommitTransaction() is called. Note that the GFT service only supports up to 500 INSERTs and up to 1MB of content per transaction.

Note : only CreateFeature() makes use of OGR transaction mechanism. SetFeature() and DeleteFeature() will still be issued immediately.

SQL

SQL commands provided to the OGRDataSource::ExecuteSQL() call are executed on the server side, unless the OGRSQL dialect is specified. The subset of SQL supported by the GFT service is described in the links at the end of this page.

The SQL supported by the server understands only natively table id, and not the table names returned by OGR. For conveniency, OGR will "patch" your SQL command to replace the table name by the table id however.

Examples

- Listing the tables and views owned by the authenticated user:

  ```
  ogrinfo -ro "GFT:email=john.doe@example.com password=secret_password"
  ```
- Creating and populating a table from a shapefile:

  ```
  ogr2ogr -f GFT "GFT:email=john.doe@example.com password=secret_password" shapefile.shp
  ```
- Displaying the content of a public table with a spatial and attribue filters:

  ```
  ogrinfo -ro "GFT:tables=224453" -al -spat 67 31.5 67.5 32 -where "'Attack on' = 'ENEMY'"
  ```
- Getting the auth key:

  ```
  ogrinfo --config CPL_DEBUG ON "GFT:email=john.doe@example.com password=secret_password"
  ```

  returns:

  ```
  HTTP: Fetch(https://www.google.com/accounts/ClientLogin)
  HTTP: These HTTP headers were set: Content-Type: application/x-www-form-urlencoded
  GFT: Auth key : A_HUGE_STRING_WITH_ALPHANUMERIC_AND_SPECIAL_CHARACTERS
  ```

  Now, you can set the GFT_AUTH environment variable to that value and simply use "GFT:" as the DSN.

See Also

- Google Fusion Tables Developer's Guide
- Google Fusion Tables Developer's Reference

GML - Geography Markup Language

OGR has limited support for GML reading and writing. Update of existing files is not supported.

Supported GML flavors :

OGR version	Read	Write
OGR >= 1.8.0	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)
OGR < 1.8.0	GML2 and limited GML3	GML 2.1.2

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 overriden by specifying the **GML_PARSER** configuration option to **EXPAT** or **XERCES**.

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:", 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.

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.

The first time a GML file is opened, if the associated .xsd is absent or could not been parsed correctly, it is completely scanned in order to determine the set of featuretypes, the attributes associated with each and other dataset level information. This information is stored in a .gfs file with the same basename as the target gml file. Subsequent accesses to the same GML file will use the .gfs file to predefine dataset level information accelerating access. To a limited extent the .gfs file can be manually edited to alter how the GML file will be parsed. Be warned that the .gfs file will be ignored if the associated .gml file has a newer timestamp.

When prescanning the GML file to determine the list of feature types, and fields, the contents of fields are scanned to try and determine the type of the field. In some applications it is easier if all fields are just treated as string fields. This can be accomplished by setting the configuration option **GML_FIELDTYPES** to the value **ALWAYS_STRING**.

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 occurences.

Since OGR 1.8.0, a specialized GML driver - the NAS driver - is available to read German AAA GML Exchange Format (NAS/ALKIS).

Configuration options can be set via the CPLSetConfigOption() function or as environment variables.

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.

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.

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 temperary file generated using CPLGenerateTempFilename("ResolvedGML"); if it cannot, resolution fails.

Note that the resolution algorithm is not optimised 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 won't be needed. Mention a comma seperated 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.

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 mentionned 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 mentionned 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.

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 overriden 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.

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 MONOBLOCK_LAYERS if not explicitely 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);
```

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:

- **XSISCHEMAURI**: If provided, this URI will be inserted as the schema location. Note that the schema file isn't actually accessed by OGR, so it is up to the user to ensure it will match the schema of the OGR produced GML data file.
- **XSISCHEMA**: 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 XSISCHEMAURI is used).
- **FORMAT**: (OGR >= 1.8.0) This can be set to GML3 in order to write GML files that follow GML3 SF-0 profile. Otherwise GML2 will be used.
- **GML3_LONGSRS**=YES/NO. (OGR >= 1.8.0, only valid when FORMAT=GML3) Default to YES. If YES, SRS with EPSG authority will be written with the "urn:ogc:def:crs:EPSG::" prefix. In the case, if the SRS is a geographic SRS without explicit AXIS order, but that the same SRS authority code imported with ImportFromEPSGA() should be treated as lat/long, 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.
- **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.

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&gt 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 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)
```

Example

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 wher
```

See Also

- GML Specifications
- GML 3.1.1 simple features profile
- Xerces
- NAS/ALKIS : specialized GML driver for cadastral data in Germany

GMT ASCII Vectors (.gmt)

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 auxilary information including geometry type, layer extents, coordinate system and attribute field declarations in comments in the header, and for each feature can have attributes.

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 seperate .gmt file.

GPSBabel

(GDAL/OGR >= 1.8.0)

The GPSBabel driver for now that relies on the GPSBabel utility to access various GPS file formats.

The GPSBabel executable must be accessible through the PATH.

Read support

The driver needs the GPX driver to be fully configured with read support (through Expat library) to be able to parse the output of GPSBabel, as GPX is used as the intermediate pivot format.

The returned layers can be waypoints, routes, route_points, tracks, track_points depending on the input data.

The syntax to specify an input datasource is :
GPSBabel:gpsbabel_file_format[,gpsbabel_format_option]:[features=[waypoints,][tracks,][routes]:]filename*
where :

- *gpsbabel_file_format* is one of the file formats handled by GPSBabel.
- *gpsbabel_format_option* is any option handled by the specified GPSBabel format (refer to the documentation of each GPSBabel format)
- *features=* can be used to modify the type of features that GPSBabel will import. waypoints matches the -w option of gpsbabel commmandline, tracks matches -t and routes matches -r. This option can be used to require full data import from GPS receivers that are slow and for which GPSBabel would only fetch waypoints by default. See the documentation on Route and Track modes for more details.
- *filename* can be an actual on-disk file, a file handled through the GDAL virtual file API, or a special device handled by GPSBabel such as "usb:", "/dev/ttyS0", "COM1:", etc.. What is actually supported depends on the used GPSBabel format.

Alternatively, for a few selected GPSBabel formats, just specifying the filename might be sufficient. The list includes for now :

- garmin_txt
- gdb
- magellan
- mapsend
- mapsource
- nmea
- osm
- ozi

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.

Write support

The driver relies on the GPX driver to create an intermediate file that will be finally translated by GPSBabel to the desired GPSBabel format. (The GPX driver does not need to be configured for read support for GPSBabel write support.).

The support geometries, options and other creation issues are the ones of the GPX driver. Please refer to its documentation for more details.

The syntax to specify an output datasource is :
GPSBabel:gpsbabel_file_format[,gpsbabel_format_option]:filename* where :

- *gpsbabel_file_format* is one of the file formats handled by GPSBabel.
- *gpsbabel_format_option* is any option handled by the specified GPSBabel format (refer to the documentation of each GPSBabel format)

Alternatively, you can just pass a filename as output datasource name and specify the dataset creation option GPSBABEL_DRIVER=gpsbabel_file_format[,gpsbabel_format_option]*

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.

Examples

- Reading the waypoints from a Garmin USB receiver :

```
ogrinfo -ro -al GPSBabel:garmin:usb:
```

- Converting a shapefile to Magellan Mapsend format :

```
ogr2ogr -f GPSBabel GPSBabel:mapsend:out.mapsend in.shp
```

See Also

- GPSBabel Home Page
- GPSBabel file formats
- GPX driver page

GPX - GPS Exchange Format

(Starting with GDAL 1.5.0)

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.

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 mentionned 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 mentionned 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.

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.

Creation Issues

On export all layers are written to a single GPX file. Update of existing files is not currently supported.

If the output file already exits, the writing will not occur. You have to delete the existing file first.

Supported geometries :

- Features of type wkbPoint/wkbPoint25D are written in the *wpt* element.
- Features of type wkbLineString/wkbLineString25D are written in the *rte* element.
- Features of type wkbMultiLineString/wkbMultiLineString25D are written in the *trk* element.
- Other type of geometries are not supported.

For route points and tracks points, if there is a Z coordinate, it is used to fill the elevation element of the corresponding points.

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 writting 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 writting 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.

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
  ```

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">
  <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"
  ```

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 explicitely specify -nlt POINT (or LINESTRING or MULTILINESTRING).

See Also

- Home page for GPX format
- GPX 1.1 format documentation

GRASS

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. wkbMulti* 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.

Datasource name

Datasource name is full path to 'head' file in GRASS vector directory. Using names of GRASS enviroment 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
```

Layer names

Usualy layer numbers are used as layer names. Layer number 0 is used for all features without any category. It is possible to optionaly 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'.

Attribute filter

If a layer has attributes stored in a database, the query is passed to the underlaying 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 offeres 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.

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.

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 enviroment variable. GISBASE is used to find GRASS database drivers.

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.

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.

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.

See Also

- GRASS home page

Development of this driver was financially supported by Faunalia (www.faunalia.it).

GTM - GPS TrackMaker

(Starting with GDAL 1.7.0)

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.

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.

See Also

- Home page for GPS TrackMaker Program
- GTM 211 format documentation

HTF - Hydrographic Transfer Format

(GDAL/OGR >= 1.8.0)

This driver reads files containg 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 metres at 95% CI (2.45) with respect to the given datum.
- *DEPTH_ACCURACY* : +/- NN.n metres 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 identifer 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 metres NNNNNNN.n.
- *NORTHING* : Grid coordinate position of the sounding in metres NNNNNNN.n.
- *DEPTH* : Reduced sounding value in metres with corrections applied as indicated in the relevant fields, soundings are positive and drying heights are negative +/-NNNN.nn metres.
- *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 metres.
- *DEEP_WATER_CORRECTION* : Value of the deep water sounding velocity applied +/- NN.nn metres.
- *VERTICAL BIAS_CORRECTION* : Value of the vertical bias applied +/- NN.nn metres. eg transducer depth correction
- *SOUND_VELOCITY* : Measured sound velocity used to process sounding in metres per second IIII.
- *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.

See Also

- HTF - Hydrographic Transfer Format home page
- HTF Technical Specification

IDB

This driver implements support for access to spatial tables in IBM Informix extended with the DataBlade spatial module.

When opening a database, it's 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.

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

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'
```

INTERLIS

OGR has support for INTERLIS reading and writing.
INTERLIS is a standard which has been especially composed in order to fulfil the requirements of modelling and the integration of geodata into contemporary and future geographic information systems. The current version is INTERLIS version 2. INTERLIS version 1 remains a Swiss standard. 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

Model

Data is read and written into transfer files which have different formats in INTERLIS 1 (.itf) and INTERLIS 2 (.xml). For using the INTERLIS model (.ili) a Java interpreter is needed at runtime and ili2c.jar (included in the Compiler for INTERLIS 2) must be in the Java path. The model file can be added to the transfer file seperated by a comma.

Some possible transformations using ogr2ogr.

- Interlis 1 -> Shape:

  ```
  ogr2ogr -f "ESRI Shapefile" shpdir ili-bsp.itf,ili-bsp.ili
  ```
- Interlis 2 -> Shape:

  ```
  ogr2ogr -f "ESRI Shapefile" shpdir RoadsExdm2ien.xml,RoadsExdm2ben.ili,RoadsExdm2ien.ili
  ```

  or without model:

  ```
  ogr2ogr -f "ESRI Shapefile" shpdir RoadsExdm2ien.xml
  ```
- Shape -> Interlis 1:

  ```
  ogr2ogr -f "Interlis 1" ili-bsp.itf Bodenbedeckung__BoFlaechen_Form.shp
  ```
- Shape -> Interlis 2:

  ```
  ogr2ogr -f "Interlis 2" LandCover.xml,RoadsExdm2ben.ili
     RoadsExdm2ben_10.Roads.LandCover.shp
  ```
- Incremental import from Interlis 1 into PostGIS:

  ```
  ogr2ogr -f PostgreSQL PG:dbname=warmerda av_fixpunkte_ohne_LFPNachfuehrung.itf,av.ili
     -lco OVERWRITE=yes
  ogr2ogr -f PostgreSQL PG:dbname=warmerda av_fixpunkte_mit_LFPNachfuehrung.itf,av.ili
     -append
  ```

Arc interpolation

INTERLIS arc geometries are converted to polygons.
The interpolation angle can be changed with the environment variable ARC_DEGREES (Default: 1 degree).

Other Notes

- More Information: http://gis.hsr.ch/wiki/OGR (german)
- Development of the OGR INTERLIS driver was supported by Swiss Federal Administration, Canton Solothurn and Canton Thurgovia.

INGRES

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, it's name should be specified in the form "@driver=ingres,dbname=*dbname*[,options]" where the options can include comma seperated items like "username=*userid*", "password=*password*", "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 user.

Examples:

```
@driver=ingres,dbname=test,userid=warmerda,password=test,tables=usa/canada
```

```
@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.

Currently the INGRES driver supports only "old style" INGRES spatial data types such as POLYGON, POINT, LINE, LONG POLYGON, LONG LINE, etc. It is anticipated in the future a newer OGC compliant set of spatial types will also be supported.

Caveats

- The spatial types CIRCLE and ICIRCLE are not currently supported for reading.
- 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.
- There is currently no support for coordinate systems.

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.
- The spatial types BOX and IBOX are not supported for creating, and when read are represented as polygons.
- The non-LONG types (such as LINE, ILINE, and POLYGON) only support a limited number of vertices. Attempts to create objects with more than the maximum possible vertices for these types will fail.
- The old ingres spatial types are very particular about geometry validation, and attempts to insert (create) features with invalid geometries will fail. Reasons for invalidity include self-intersection of linestrings, and self-intersections for polygons.

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.

KML - Keyhole Markup Language

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)*

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 ouput. Since GDAL 1.6.1, folders containing multiple geometry types, like POINT and POLYGON, are supported.

KML Writing

Since not all features of KML are able to be represented in the Simple Features geometry model, you will not be able to generate many KML-specific attributes from within GDAL/OGR. Please try a few test files to get a sense of what is possible.

When outputting KML, the OGR KML driver will translate each OGR Layer into a KML Folder. (you may encounter unexpected behavior if you try to mix the geometry types of elements in a layer, e.g. `LINESTRING` and `POINT` data.

The KML Driver will rename some layers, or source KML folder names, into new names it considers valid, for example '`Layer #0`', the default name of the first unnamed Layer, becomes `'Layer__0'`.

KML is mix of formatting and feature data. The <description> tag of a Placemark will be displayed in most geobrowsers as an HTML-filled balloon. When writing KML, Layer element attributes are added as simple schema fields. This best preserves feature type information.

Limited support is available for fills, line color and other styling attributes. Please try a few sample files to get a better sense of actual behavior.

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 mentionned 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 mentionned 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.

Creation Options

The following creation options are supported:

- **NameField**: Allows you to specify the field to use for the KML <name> element. Default value : 'Name'

  ```
  ogr2ogr -f KML output.kml input.shp -dsco NameField=RegionName
  ```

- **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
  ```

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
```

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.

See Also

- KML Specification
- KML Tutorial

LIBKML Driver (.kml .kmz)

The LIBKML driver is a client of Libkml from Google, 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 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 ouput 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.

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 conatains a <NetworkLink> to each layer file in the datasource. This feature can be turned off by setting the enviroment variable LIBKML_USE_DOC.KML to "no"

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 OGR Feature Style can translate into KML.

Layer

Layers are mapped to kml files as a <Document> or <Folder>, and in kmz files or directorys as a seperate kml file.

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.

Schema

Read and write of <Schema> is supported for .kml files , .kmz files, and directorys.

Feature

An OGR feature translates to kml as a <Placemark>.

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 enviroment 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 enviroment 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 copyed to the style string.

Fields

OGR fields (feature atributes) are mapped to kml with <Schema&gt; and <SimpleData>, except for some special fields as noted below.

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 enviroment 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 enviroment 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 enviroment 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 enviroment 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 enviroment 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 enviroment 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 enviroment 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 enviroment 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 enviroment variable LIBKML_VISIBILITY_FIELD .

OGR_STYLE
: This string feild maps to a features style string, OGR reads this field if there is no style string set on the feature.

Geometry

Translation of OGR Geometry to KML Geometry is pretty strait forwards with only a couple of exceptions. Point to <Point>, 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 enviroment variable LIBKML_WRAPDATELINE to "yes" will cause the libkml driver to split the geometry at the dateline when read.

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.


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

}


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
```

Access MDB databases

GDAL/OGR >= 1.9.0

OGR optionally supports reading access .mdb files by using the Java Jackcess library.

This driver is primarly 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.

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
```

It is possible to add the *--with-jvm-lib-add-rpath* option to embed the path to the libjvm.so in the GDAL library.

On others Linux flavours, 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

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-1.2.2.jar, 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 explicitely point them with the CLASSPATH environment variable.

Resources

- Jackcess library home page
- Utility that contains the needed JARs dependencies

See also

- PGeo driver page
- Geomedia driver page

Memory

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).

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.

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.

New fields can be added to layer that already has features.

MapInfo TAB and MIF/MID

MapInfo datasets in native (TAB) format and in interchange (MIF/MID) format are supported for reading and writing. Update of existing files is not currently 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.

Creation Issues

The TAB File format requires that the bounds (geographical extents) of a new file be set before writing the first feature. However, there is currently no clean mechanism to set the default bounds of a new file through the OGRDataSource interface.

We should fix the driver at some point to set valid default bounds for each projection, but 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, -15000000) (30000000, 15000000)

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.

Dataset Creation Options

- **FORMAT=MIF**: To create MIF/MID instead of TAB files (TAB is the default).

Layer Creation Options

- **SPATIAL_INDEX_MODE=QUICK**: Use this to turn on "quick spatial index mode". The default behavior of MITAB since GDAL v1.5.0 is to generate an optimized spatial index, but this results in slower write speed than what we used to get with GDAL 1.4.x and older. Applications that want faster write speed and do not care about the performance of spatial queries on the resulting file can use this option to require the creation of a non-optimal spatial index (actually emulating the type of spatial index produced by OGR's TAB driver before GDAL 1.5.0). In this mode writing files can be about 5 times faster, but spatial queries can be up to 30 times slower.

Compatability

Before v1.8.0 , the driver was incorrectly using a "." as the delimiter for id: parameters and starting with v1.8.0 the driver uses a comma as the delimiter was per the OGR Feature Style Specification.

See Also

- MITAB Page

MSSQLSpatial - Microsoft SQL Server Spatial Database

This driver implements support for access to spatial tables in Microsoft SQL Server 2008+ which contains the geometry and geography data types to repesent the geometry columns.

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**: 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 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.

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::StartTrasaction(), OGRLayer::CommitTransaction() and OGRLayer::RollbackTransaction() calls in the normal SQL sense.

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.

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 referece 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.
- **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. 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 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
```

Examples

Creating a layer from an OGR data source

```
ogr2ogr -overwrite -f MSSQLSpatial "MSSQL:server=.\MSSQLSERVER2008;database=geodb;
   trusted_connection=yes" "rivers.tab"
```

Connecting to a layer and dump the contents

```
ogrinfo -al "MSSQL:server=.\MSSQLSERVER2008;database=geodb;tables=rivers;
   trusted_connection=yes"
```

Creating a spatial index

```
ogrinfo -sql "create spatial index on rivers" "MSSQL:server=.\MSSQLSERVER2008;
   database=geodb;trusted_connection=yes"
```

MySQL

This driver implements read and write access for spatial data in MySQL tables. This functionality was introduced in GDAL/OGR 1.3.2.

When opening a database, it's name should be specified in the form "MYSQL:dbname[,options]" where the options can include comma seperated 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.

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.
- 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.

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.

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".
- **MYSQL_GEOM_COLUMN**: This option specifies the name of the geometry column. The default value is "SHAPE".
- **MYSQL_FID**: This option specifies the name of the FID column. The default value is "OGR_FID"
- **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)
```

NAS - ALKIS

The NAS reader 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.

This driver was implemented within the context of the PostNAS project which has more information on it's use.

See Also

- PostNAS

UK .NTF

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.

See Also

- General UK NTF Information

Implementation Notes

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
        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.

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 (ie. POINT) of that product family (ie. 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.

Special Attributes

```
FEAT_CODE: General feature code integer, can be used to lookup a name in the
           FEATURE_CLASSES layer/table.

TEXT_ID/POINT_ID/LINE_ID/NAME_ID/COLL_ID/POLY_ID/GEOM_ID:
          Unique identifier for a feature of the appropriate type.

TILE_REF: All layers (except FEATURE_CLASSES) contain a TILE_REF attribute
          which indicates which tile (file) the features came from.  Generally
          speaking the id numbers are only unique within the tile and so
          the TILE_REF can be used restrict id links within features from
          the same file.

FONT/TEXT_HT/DIG_POSTN/ORIENT:
        Detailed information on the font, text height, digitizing position,
        and orientation of text or name objects.  Review the OS product
        manuals to understand the units, and meaning of these codes.
```

```
GEOM_ID_OF_POINT:
        For _NODE features this defines the POINT_ID of the point layer object
        to which this node corresponds.  Generally speaking the nodes don't
        carry a geometry of their own.  The node must be related to a point
        to establish it's 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 it's 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.
```

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".

Oracle Spatial

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, it's 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 accessable 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.

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'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.

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 (ie. 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.

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 concequence 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.

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.
- **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.
- **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. If not set 3 is used by default.
- **INDEX**: This may be set to OFF 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.
- **INDEX_PARAMETERS**: This may be set to pass creation parameters when the spatial index is created. For instance setting INDEX_PARAMETERS to SDO_LEVEL=5 would cause a 5 level tile index to be used. By default no parameters are passed causing a default R-Tree spatial index to be created.
- **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 Y 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 it if 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.
- **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.

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'"
```

Credits

I would like to thank SRC, LLC for it's financial support of the development of this driver.

ODBC RDBMS

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 seperated 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 timeconsuming, 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.

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.

See Also

- MSDN ODBC API Reference

OGDI Vectors

OGDI vector support is optional in OGR, and is normally only configured if OGDI is installed on the build system. If available OGDI 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 OGDI gltp url is opened directly the OGDI 3.1 capabilities for the driver/server are queried to get a list of layers. One OGR layer is created for each OGDI family available for each layer in the datastore. For drivers such as VRF this can result in alot of layers. Each of the layers has an OGR name based on the OGDI 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_OGDI_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 OGDI 3.1 drivers as before OGDI 3.1 there was no regular way to discover available layers in OGDI.

```
gltp:[//<hostname>]/<driver_name>/<dataset_name>:<layer_name>:<family>
```

Where <layer_name> is the OGDI Layer name, and <family> is one of: "line", "area", "point", or "text".

OGDI 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 OGDI driver.

Raster layers cannot be accessed with this driver but can be accessed using the GDAL OGDI Raster driver.

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 ing of the layer name.

```
ogrinfo -ro gltp:/vrf/usr4/mpp1/v0noa/vmaplv0/noamer \
        -sql 'select * from "polbndl@bnd(*)_line" where use=26'
```

See Also

- OGDI.SourceForge.Net
- VMap0 Coverages

OpenAir - OpenAir Special Use Airspace Format

(GDAL/OGR >= 1.8.0)

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 tesselated. 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.

See Also

- Description of OpenAir format

PDS - Planetary Data Systems TABLE

(GDAL/OGR >= 1.8.0)

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.

See Also

- Description of PDS format (see Annex A.29 from StdRef_20090227_v3.8.pdf)

PostgreSQL / PostGIS

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 (GDAL/OGR >= 1.8.0)

You can find additionnal information on the driver in the Advanced OGR PostgreSQL driver Information page.

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'"
```

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.

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.

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::StartTrasaction(), OGRDataSource::CommitTransaction() and OGRDataSource::RollbackTransaction() calls in the normal SQL sense.

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 (ie. 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 catched by enclosing this command with a CPLPushErrorHandler()/CPLPopErrorHandler() pair.

Dataset Creation Options

None

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}**: Control the dimension of the layer. Defaults to 3. 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**: (From GDAL 1.6.0) Set to ON by default. Creates a spatial index on the geometry column to speed up queries. Set to OFF to disable. (Has effect only when PostGIS is available).
- **TEMPORARY**: (From GDAL 1.8.0) Set to OFF by default. Creates a temporary table instead of a permanent one.
- **NONE_AS_UNKNOWN**: (From GDAL 1.8.1) Can bet 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'.
- **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.

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 less robust than INSERT, but significantly faster.
- **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 transfered 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.

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 OGDI 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'
  ```
- 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 = 'Albe
  ```
- 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'
  ```

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)

See Also

- Advanced OGR PostgreSQL driver Information
- OGR PostgreSQL SQL Dump driver Page
- PostgreSQL Home Page
- PostGIS
- PostGIS / OGR Wiki Examples Page

PostgreSQL SQL Dump

(GDAL/OGR >= 1.8.0)

This write-only driver implements support for generating a SQL dump file that can later be injected into a live PostgreSQL instance. It supports PostgreSQL extended with the PostGIS geometries.

This driver is very similar to the PostGIS shp2pgsql utility.

Most creation options are shared with the regular PostgreSQL driver.

The driver opens the output file through the VSIF Large API, so it is possible to specify /vsistdout/ as output file to output on the standard output.

Creation options

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).

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}**: Control the dimension of the layer. Defaults to 3. 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**: Set to ON by default. Creates a spatial index on the geometry column to speed up queries. Set to OFF to disable. (Has effect only when PostGIS is available).
- **TEMPORARY**: Set to OFF by default. Creates a temporary table instead of a permanent one.
- **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**: (OGR >= 1.8.1) Set to ON by default so that tables are destroyed before being recreated. Set to OFF to prevent DROP TABLE from being emitted. Set to IF_EXISTS in order DROP TABLE IF EXISTS to be emitted (needs PostgreSQL >= 8.2)
- **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 mentionned, 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 bet 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'.
- **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.

Environment variables

- **PG_USE_COPY**: This may be "YES" for using COPY for inserting data to Postgresql. COPY is less robust than INSERT, but significantly faster.

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 peformance.

```
% 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 -
```

See Also

- OGR PostgreSQL driver Page
- PostgreSQL Home Page
- PostGIS
- PostGIS / OGR Wiki Examples Page

ESRI Personal GeoDatabase

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.

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

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)

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
CPTimeout  =
CPReuse    =
```

- [Microsoft Access Driver (*.mdb)] - remember to use "Microsoft Access Driver (*.mdb)" as the name of section because PGeo driver composes ODBC connection string for Personal Geodatabase using "DRIVER=Microsoft Access Driver (*.mdb);" string.
- Description - put short description of this driver definition.
- Driver - full path of ODBC driver for MDB Tools.

2. ODBC data source configuration

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.

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: ...
```

Resources

- About ESRI Geodatabase
- [mdbtools-dev] DSN-less connection not supported?

See also

- MDB driver page

IHO S-57 (ENC)

IHO S-57 datasets are supported for read access.

The S-57 driver module produces features for all S-57 features in S-57 file, and associated updates. S-57 (ENC) files normally have the extension ".000".

S-57 feature objects are translated into features. S-57 geometry objects are automatically collected and formed into geometries on the features.

The S-57 reader depends on having two supporting files, s57objectclasses.csv, and s57attributes.csv available at runtime in order to translate features in an object class specific manner. These should be in the directory pointed to by the environment variable S57_CSV, or in the current working directory.

S-57 update files contain information on how to update a distributed S-57 base data file. The base files normally have the extension .000 while the update files have extensions like .001, .002 and so on. The S-57 reader will normally read and apply all updates files to the in memory version of the base file on the fly. The feature data provided to the application therefore includes all the updates.

Feature Translation

Normally all features read from S-57 are assigned to a layer based on the name of the object class (OBJL) to which they belong. For instance, with an OBJL value of 2, the feature is an "Airport / airfield" and has a short name of "AIRARE" which is used as the layer name. A typical S-57 transfer will have in excess of 100 layers.

Each feature type has a predefined set of attributes as defined by the S-57 standard. For instance, the airport (AIRARE) object class can have the AIRARE, CATAIR, CONDTN, CONVIS, NOBJNM, OBJNAM, STATUS, INFORM, NINFOM, NTXTDS, PICREP, SCAMAX, SCAMIN, TXTDSC, ,RECDAT, RECIND, SORDAT, and SORIND attributes. These short names can be related to longer, more meaningful names using an S-57 object/attribute catalog such as the S-57 standard document itself, or the catalog files (s57attributes.csv, and s57objectclasses.csv). Such a catalog can also be used to establish all the available object classes, and their attributes.

The following are some common attributes, including generic attributes which appear on all feature, regardless of object class. is turned on.

```
Attribute Name  Description                            Defined On
--------------  -----------                            ----------

GRUP            Group number.                          All features

OBJL            Object label code.  This number        All features
                indicates the object class of the
                feature.

RVER            Record version.

AGEN            Numeric agency code, such as 50 for    All features
                the Canadian Hydrographic Service.
                A potentially outdated list is
                available in agencode.txt.

FIDN            Feature identification number.         All features

FIDS            Feature identification subdivision.    All features

DSNM            Dataset name.  The name of the file    All features
                the feature came from.  Used with
```

	LNAM to form a unique dataset wide identifier for a feature.	
INFORM	Informational text.	Some features
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

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 it's 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.

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.

- **UPDATES**=APPLY/IGNORE: Should update files be incorporated into the base data on the fly. Default is APPLY.
- **SPLIT_MULITPOINT**=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_MULTIPOINT 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 OFF.

- **RETURN_LINKAGES**=ON/OFF: Should additional attributes relating features to their underlying geometric primtives 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.

Example:

```
set OGR_S57_OPTIONS = "RETURN_PRIMITIVES=ON,RETURN_LINKAGES=ON,LNAM_REFS=ON"
```

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"
```

See Also

- S-57 Online Object/Attribute Catalog
- Frank's S-57 Page: Links to other resources, and sample datasts.

ESRI ArcSDE

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 tablenames. 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.

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.

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 reconsile 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.
-

Examples

See the ogr_sde.py test script for some example connection strings and usage of the driver.

SDTS

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.

See Also

- SDTS Abstraction Library: The base library used to implement this driver.
- http://mcmcweb.er.usgs.gov/sdts: Main USGS SDTS web page.

ESRI Shapefile

All varieties of ESRI Shapefiles should be available for reading, and simple 3D files can be created.

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.

The new ESRI measure values are discarded if encountered. MultiPatch files are read and each patch geometry is turned into a multi-polygon representation with one polygon per triangle in triangle 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.

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.

An attempt is made to read 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 ISO8859_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. (Recoding support is new for GDAL/OGR 1.9.0)

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. It does not support the ESRI spatial index files (.sbn / .sbx). Spatial indexing can accelerate spatially filtered passes through large datasets to pick out a small area quite dramatically.

To create a spatial index, 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.

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. Longer names will be silently truncated. This may result in non-unique column names, which will definitely cause problems later.
- 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_1
- Only Integer, 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 11.
- 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.

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.

Size Issues

Geometry: The Shapefile format explicitly uses 32bit offsets and so cannot go over 8GB (it actually uses 32bit offsets to 16bit words). Hence, it is is not recommended to use a file size over 4GB.

Attributes: The dbf format does not have any offsets in it, so it can be arbitrarily large.

Dataset Creation Options

None

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, or POINTZ, ARCZ, POLYGONZ or MULTIPOINTZ for 3D. Shapefiles with *measure* values are not supported, nor are MULTIPATCH files.
- **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.

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.

See Also

- Shapelib Page
- User Notes on OGR Shapefile Driver

SQLite RDBMS

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 respectible performance.

The driver looks for a geometry_columns table layed 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.

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. None of the list attribute field types existing in SQLite.

SQLite databases often due 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.

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. Attributes queries may be fast, especially if indexes are built for appropriate attribute columns using the "CREATE INDEX ON ( )" SQL command.

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.

Starting with OGR 1.8.0, 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.

Using the SpatiaLite library (Spatial extension for SQLite)

(Starting with GDAL 1.7.0)

The SQLite driver can read and write SpatiaLite databases. Updating a spatialite database requires explicit linking against SpatiaLite library (version >= 2.3). 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
```

Creation Issues

The SQLite driver supports creating new SQLite database files, or adding tables to existing ones. Note that a new database file cannot be created over an existing file.

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 flavour of the metadata tables, which are a bit differ from the metadata used by this OGR driver and from OGC specifications. Implies **METADATA=yes**.
- **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.

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
- **LAUNDER=yes/no**: Controls whether layer and field names will be laundered for easier use in SQLite. Laundered names will be convered to lower case and some special characters will be changed to underscores.
- **SPATIAL_INDEX=yes/no**: (Starting with GDAL 1.7.0) If the database is of the SpatiaLite flavour, and if OGR is linked against libspatialite, this option can be used to control if a spatial index must be created. Default to yes.

Other Notes

- Development of the OGR SQLite driver was supported by DM Solutions Group and GoMOOS.
- http://www.sqlite.org: Main SQLite page.
- http://www.gaia-gis.it/spatialite/: SpatiaLite extension to SQLite.
- FDO RFC 16: FDO Provider for SQLite

SUA - Tim Newport-Peace's Special Use Airspace Format

(GDAL/OGR >= 1.8.0)

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 tesselated.

See Also

- Description of .SUA format

SVG - Scalable Vector Graphics

(OGR >= 1.9.0)

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

See Also

- W3C SVG page
- Cloudmade vector documentation

U.S. Census TIGER/Line

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 equivelents. Each county consists of a series of files with a common basename, and different extentions. For instance, county 1 in state 26 (Michigan) consists of the following file set in Tiger98.

```
TGR26001.RT1
TGR26001.RT2
TGR26001.RT3
TGR26001.RT4
TGR26001.RT5
TGR26001.RT6
TGR26001.RT7
TGR26001.RT8
TGR26001.RT9
TGR26001.RTA
TGR26001.RTC
TGR26001.RTH
TGR26001.RTI
TGR26001.RTP
TGR26001.RTR
TGR26001.RTS
TGR26001.RTZ
```

The TIGER/Line coordinate system is hardcoded to NAD83 lat/long degrees. This should be appropriate for all recent years of TIGER/Line production.

There is no update or creation support in the TIGER/Line driver.

The reader was implemented for TIGER/Line 1998 files, but some effort has gone into ensuring compatibility with 1992, 1995, 1997, 1999, 2000, 2002, 2003 and 2004 TIGER/Line products as well. The 2005 products have also been reported to work fine. It is believe that any TIGER/Line product from the 1990's should work with the reader, with the possible loss of some version specific information.

Feature Representation

With a few exceptions, a feature is created for each record of a TIGER/Line data file. Each file (ie. .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.

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's 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.

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 altername 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.

FeatureIds

These features are derived from type 5 (Complete Chain Feature Identifiers) records. Each feature contains a feature name (FENAME), and it's 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.

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.

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.

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.

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.

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.

EntityNames

These features are derived from type C (Geographic Entity Names) records.

These features have no associated geometry.

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.

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.

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.

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.

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.

VFK - Czech cadastral exchange data format

VFK driver can read data in the *Czech cadastral exchange data format*. The VFK file is recognized as an OGR datasource with zero or more OGR layers.

Points are represented as wkbPoints, lines and boundaries as wkbLineStrings and areas as wkbPolygons. wkbMulti* primitives are not used. Feature types cannot be mixed in one layer.

Datasource name

Datasource name is a full path to the VFK file.

Layer names

VFK data blocks are used as layer names.

Attribute filter

OGR internal SQL engine is used to evaluate the expression. Evaluation is done once when the attribute filter is set.

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.

References

- OGR VFK Driver Implementation Issues
- Czech cadastral exchange data format documentation (in Czech)

Virtual Format

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.

Creation Issues

Before GDAL 1.7.0, the OGR VRT driver was read-only.

Since GDAL 1.7.0, 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

Virtual File Format

The root element of the XML control file is **OGRVRTDataSource**. It has an **OGRVRTLayer** child for each layer in the virtual datasource. That 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.
- **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.
- **FID** (optional): Name of the attribute column from which the FID of features should be derived. If not provided, the FID of the source features will be used directly.
- **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** and **z** attributes will have the names of the columns to be used for the X, Y and Z coordinates. The **z** attribute is optional.
  Starting with GDAL 1.7.0, the optional **reportSrcColumn** attribute can be used to specify whether the source geometry fields (the fields set in the **field**, **x**, **x** or **z** 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.
- **SrcRegion** (optionnal, 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 explicitely 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".
  - **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. By default defaults to the value of "name".

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>
```

Example: Renaming attributes

It can be usefull 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&gt, 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</SrcSQL>
        <Field name="name" src="src_field_1" />
        <Field name="desc" src="src_field_2" type="String" width="45" />
    </OGRVRTLayer>
</OGRVRTDataSource>
```

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>
```

Other Notes

- When the *GeometryField* is "WKT" spatial filtering is applied after extracting all rows from the source datasource. Essentially that means there is no fast spatial filtering on WKT derived geometries.
- When the *GeometryField* is "PointFromColumns", and a *SrcLayer* (as opposed to *SrcSQL*) is used, and a spatial filter is in effect on the virtual layer then the spatial filter will be internally translated into an attribute filter on the X and Y columns in the *SrcLayer*. In cases where fast spatial filtering is important it can be helpful to index the X and Y columns in the source datastore, if that is possible. For instance if the source is an RDBMS. You can turn off that feature by setting the *useSpatialSubquery* attribute of the GeometryField element to FALSE.
- Normally the *SrcDataSource* is a non-spatial tabular format (such as MySQL, SQLite, CSV, OCI, or ODBC) but it can also be a spatial database in which case the geometry can be directly copied over.

WFS - OGC WFS service

(GDAL/OGR >= 1.8.0)

This driver can connect to a OGC WFS service. It supports WFS 1.0.0 and WFS 1.1.0 protocols. GDAL/OGR must be built with Curl support in order to 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 Transactionnal ones (WFS-T).

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*

Additionnal optional parameters can be specified such as *TYPENAME*, *VERSION*, *MAXFEATURES* as specified in WFS specification.

It is also possible to specify the name of an XML file whose content matches the following syntax (the <OGRWFSDataSource&gt element must be the first bytes of the file):

```
<OGRWFSDataSource>
    <URL>http://path/to/WFS/service[?OPTIONAL_PARAMETER1=VALUE[OPTIONNAL_PARAMETER2=VALUE]]</URL>
</OGRWFSDataSource>
```

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.

Request paging

Generally, 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 a vendor specific option, STARTINDEX, that allow to do the requests per "page", and thus to avoid downloading the whole content of the layer in a single request. 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. Those 2 options can also be set in a WFS XML description file.

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.

Write support / WFS-T

The WFS-T protocol only eanbles 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 caveouts 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 successfull, 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().

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 attribue gml_id as the count of inserted features during the last commited transaction.

Note : currently, only CreateFeature() makes use of OGR transaction mechanism. SetFeature() and DeleteFeature() will still be issued immediately.

Special SQL commands

The following SQL / pseudo-SQL commands passed to OGRDataSource::ExecuteSQL() are specific of the WFS driver :

- "DELETE FROM layer_name WHERE expression" : this will result into a WFS delete operation. This can be a fast way of deleting one or several features. In particularly, this can be a faster replacement for OGRLayer::DeleteFeature() when the gml:id is known, but the feature has not been fetched from the server.
- "SELECT _LAST_INSERTED_FIDS_ FROM layer_name" : see above paragraph.

Currently, any other SQL command will be processed by the generic layer, meaning client-side only processing. Server side spatial and attribute filtering must be done through the SetSpatialFilter() and SetAttributeFilter() interfaces.

Layer metadata

(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").

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
```

See Also

- OGC WFS Standard
- GML driver documentation

X-Plane/Flightgear aeronautical data

The X-Plane aeronautical data is supported for read access. This data is for example used by the X-Plane and Flighgear software.

The driver is able to read the following files :

Filename	Description	Supported versions
apt.dat	Airport data	850, 810
nav.dat (or earth_nav.dat)	Navigation aids	810, 740
fix.dat (or earth_fix.dat)	IFR intersections	600
awy.dat (or earth_awy.dat)	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 specificiation. 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), altough the specificiation 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.

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 PostreSQL database :

```
% PG_USE_COPY=yes ogr2ogr -overwrite -f PostgreSQL PG:"dbname=apt" apt.dat
```

See Also

- X-Plane data file definitions

Airport data (apt.dat)

This file contains the description of elements defining airports, heliports, seabases, 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.

APT layer

Main description for an aiport. 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 (4.0). 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.

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 (4.0). 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 aiport.
- 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.

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 (4.0)

- 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.

WaterRunwayThreshold (Point)

Fields:

- apt_icao: String (4.0)
- rwy_num: String (3.0). Code for the runway, such as 18. Unique for each aiport.
- 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.

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 (4.0)
- 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)

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 (4.0)
- 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.

Helipad (Point)

This layer contains the center of a helipad. Fields:

- apt_icao: String (4.0)

- helipad_name: String (5.0). Name of the helipad in the format "Hxx". Unique for each aiport.
- 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)

HelipadPolygon (Polygon)

This layer contains the rectangular shape of a helipad. The fields are identical to the Helipad layer.

TaxiwayRectangle (Polygon) - V810 record

This layer contains the rectangular shape of a taxiway. Fields:

- apt_icao: String (4.0)
- 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.

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 (4.0)
- 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

APTBoundary (Polygon)

This layer contains the boundary of the aiport. There is at the maximum one such feature per aiport. The polygon may include holes. Bezier curves are discretized into linear pieces.

Fields:

- apt_icao: String (4.0)

- name: String (0.0)

APTLinearFeature (Line String)

This layer contains linear features. Bezier curves are discretized into linear pieces.

Fields:

- apt_icao: String (4.0)
- name: String (0.0)

StartupLocation (Point)

Define gate positions, ramp locations etc.

Fields:

- apt_icao: String (4.0)
- name: String (0.0)
- true_heading_deg: Real (6.2)

APTLightBeacon (Point)

Define airport light beacons.

Fields:

- apt_icao: String (4.0)
- 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

APTWindsock (Point)

Define airport windsocks.

Fields:

- apt_icao: String (4.0)
- name: String (0.0)
- is_illuminated: Integer (1.0)

TaxiwaySign (Point)

Define airport taxiway signs.

Fields:

- apt_icao: String (4.0)

- text: String (0.0). This is somehow encoded into a specific format. See X-Plane specification for more details.
- true_heading_deg: Real (6.2)
- size: Integer (1.0). From 1 to 5. See X-Plane specification for more details.

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 (4.0)
- 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)

ATCFreq (None)

Define an airport ATC frequency. Note that this layer has no geometry.

Fields:

- apt_icao: String (4.0)
- 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.

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)

ILS (Point)

Localiser that is part of a full ILS, or Stand-alone localiser (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 (4.0). 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 localiser in degree.

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.

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)

GS - Glideslope (Point)

Glideslope nav-aid.

Fields :

- navaid_id: String (4.0). Identification of nav-aid. *NOT* unique.
- apt_icao: String (4.0). 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)

Marker - ILS marker beacons. (Point)

Nav-aids of type Outer Marker (OM), Middle Marker (MM) or Inner Marker (IM).

Fields:

- apt_icao: String (4.0). 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.

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

DMEILS (Point)

DME element of an ILS.

Fields:

- navaid_id: String (4.0). Identification of nav-aid. *NOT* unique.
- apt_icao: String (4.0). 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

IFR intersections (fix.dat)

This file contain IFR intersections (often referred to as "fixes").

The following layer is reported :

- FIX (Point)

FIX (Point)

Fields:

- fix_name: String (5.0). Intersection name. *NOT* unique.

Airways (awy.dat)

This file contains the description of airway segments.

The following layers are reported :

- AirwaySegment (Line String)
- AirwayIntersection (Point)

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) : Fligh level (hundreds of feet) of the base of the airway.
- top_FL: Integer (3.0) : Fligh level (hundreds of feet) of the top of the airway.

AirwayIntersection (Point)

Fields:

- name: String (0.0) : Name of intersection or nav-aid

OGR SQL

The OGRDataSource supports executing commands against a datasource via the OGRDataSource::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.

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.

SELECT

The SELECT statement is used to fetch layer features (analygous 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 analygous to tables in an RDBMS and feature attributes are analygous to column values. The simpliest 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 equivelent 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
```

Field 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 alot 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 (numericla sum), MIN (lexical or numerical minimum), and MAX (lexical or numerical maximum). This example produces a variety of sumarization 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"
```

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 though it would get the same result as giving it any of the column names. 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
```

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.

```
SELECT cost+tax from invoice
```

or

```
SELECT CONCAT(owner_first_name,' ',owner_last_name) from properties
```

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:

1. character(field_length). By default, field_length=1.

2. float(field_length)
3. numeric(field_length, field_precision)
4. integer(field_length)
5. date(field_length)
6. time(field_length)
7. timestamp(field_length)

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.

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 explanitory, but is is worth nothing that **!=** is the same as **<>**, the string equality is case insensitive, but the **<**, **>**, **<=** and **>=** operators *are* case sensitive. Both the LIKE and ILIKE operators are case insensitive.

The value argument to the **LIKE** operator 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 it's 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 precidence 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

1. Fields must all come from the primary table (the one listed in the FROM clause).
2. 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.

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 asterix 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 untill 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 ables 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
```

JOIN Limitations

1. Joins can be very expensive operations if the secondary table is not indexed on the key field being used.
2. 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.
3. 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.
4. Datasource names for joined tables are evaluated relative to the current processes working directory, not the path to the primary datasource.
5. 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.

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 pecedence 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

(Since GDAL 1.7.0)

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%'
```

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

1. Indexes are not maintained dynamically when new features are added to or removed from a layer.
2. Very long strings (longer than 256 characters?) cannot currently be indexed.
3. To recreate an index it is necessary to drop all indexes on a layer and then recreate all the indexes.
4. Indexes are not used in any complex queries. Currently the only query the will accelerate is a simple "field = value" query.

DROP INDEX

The OGR SQL DROP INDEX command can be used to drop all indexes on a particular table, or just the index for a particular column.

```
DROP INDEX ON nation USING nation_id
DROP INDEX ON nation
```

ExecuteSQL()

SQL is executed against an OGRDataSource, not against a specific layer. The call looks like this:

```
OGRLayer * OGRDataSource::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 OGRDataSource::ReleaseResultsSet() method when no longer needed. Failure to release it before the datasource is destroyed may result in a crash.

Non-OGR SQL

All OGR drivers for database systems: MySQL, PostgreSQL and PostGIS (PG), Oracle (OCI), SQLite, ODBC, ESRI Personal Geodatabase (PGeo) and MS SQL Spatial (MSSQLSpatial), override the OGRDataSource::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.
