Reference Manual

Contents

Chapter 1

GDAL - Geospatial Data Abstraction Library

Select language: [English] [Russian] [Portuguese] [French/Francais]

GDAL is a translator library for raster geospatial data formats that is released under an X/MIT style Open Source license by the Open Source Geospatial Foundation. As a library, it presents a single abstract data model to the calling application for all supported formats. It also comes with a variety of useful commandline utilities for data translation and processing. The NEWS page describes the October 2008 GDAL/OGR 1.5.3 release.

The related OGR library (which lives within the GDAL source tree) provides a similar capability for simple features vector data.

Master: http://www.gdal.org

Download: ftp at remotesensing.org, http at download.osgeo.org

1.1 User Oriented Documentation

- Wiki - Various user and developer contributed documentation and hints
- Downloads - Ready to use binaries (executables)
- Supported Formats
- GDAL Utility Programs
- GDAL FAQ
- GDAL Data Model
- GDAL/OGR Governance and Community Participation
- GDAL Service Provider Listings (not vetted)
- Sponsors, Acknowledgements and Credits
- Software Using GDAL

1.2 Developer Oriented Documentation

- Building GDAL From Source
- Downloads - source code
- API Reference Documentation
- GDAL API Tutorial
- GDAL Driver Implementation Tutorial
- GDAL Warp API Tutorial
- OGRSpatialReference Tutorial
- GDAL C API
- GDALDataset
- GDALRasterBand
- GDAL for Windows CE

1.3 Mailing List

A gdal-announce mailing list subscription is a low volume way of keeping track of major developments with the GDAL/OGR project.

The gdal-dev@lists.osgeo.org mailing list can be used for discussion of development and user issues related to GDAL and related technologies. Subscriptions can be done, and archives reviewed on the web. The mailing list is also available in read-only format by NNTP at news://news.gmane.org/gmane.comp.gis.gdal.devel and by HTTP at http://news.gmane.org/gmane.comp.gis.gdal.devel.

Some GDAL/OGR users and developers can also often be found in the gdal IRC channel on irc.freenode.net.

1.4 Bug Reporting

GDAL bugs can be reported, and can be listed using Trac.

1.5 GDAL In Other Languages

The following bindings of GDAL in other languages are available:

- Perl
- Python
- VB6 Bindings (not using SWIG)
- GDAL Bindings into R by Timothy H. Keitt.

- Ruby
- Java
- C# / .Net

Chapter 2

Sponsors, Acknowledgements and Credits

There are too many people who have helped since GDAL/OGR was launched in late 1998 for me to thank them all. I have received moral support, financial support, code contributions, sample datasets, and bug reports from literally hundreds of people. However, below I would like to single out a few people and organizations who have supported GDAL over the years. Forgive me for all those I left out.

Frank Warmerdam

2.1 Sponsorship

Sponsors help fund maintenance, development and promotion of GDAL/OGR. If your organization depends on GDAL/OGR consider becoming a sponsor.

2.1.1 Silver Sponsors

2.1.2 Other Sponsors

- MicroImages Inc.

2.2 Personal

- **Andrey Kiselev**: my right hand man on GDAL for several years. He is primarily responsible for the HDF, MrSID, L1B, and PCIDSK drivers. He has also relieved me of most libtiff maintenance work.
- **Daniel Morissette**: for his key contributions to CPL library, and development of the Mapinfo TAB translator.
- **Howard Butler**: for substantial improvements to the python bindings.
- **Ken Shih**: for the bulk of the implementation of the OLE DB provider.
- **Markus Neteler**: for various contributions to GDAL documentation and general supportiveness.
- **Silke Reimer**: for work on Debian, and RPM packaging as well as the GDAL man pages.
- **Alessandro Amici**: for work on configuration and build system, and for the initial Debian packaging.
- **Stephane Villeneuve**: for development of the Mapinfo MIF translator.
- **Marin Byrne**: for producing the current GDAL icon set (based on the earlier version by Martin Daly).
- Darek Krawczyk: for producing design of the GDAL Team Member t-shirt (based on Marin's and Martin's graphics).

2.3 Corporate

- Applied Coherent Technologies: Supported implementation of the GDAL contour generator, as well as various improvements to HDF drivers.
- Atlantis Scientific: Supported the development of the CEOS, and a variety of other radar oriented format drivers as well as development of OpenEV, my day-to-day GDAL image viewer.

- A.U.G. Signals: Supported work on the HDF, NITF and ODBC drivers.
- Avenza Systems: Supported development of dgnlib, the basis of OGR dgn support, as well as preliminary work on image warping in GDAL.
- Cadcorp: Supported development of the Virtual Warped Raster capability.
- DM Solutions Group: Supported the development of the DGN driver, the OGR Arc/Info Binary Coverage driver, OGR WCTS (Web Coordinate Transformation Server), OGR VRT driver, ODBC driver, MySQL driver, SQLite driver, OGR JOIN and OGR C API.
- ERMapper: provided primary sponsorship for GDAL from February 2005 to November 2006 to support work on GDAL improvement efforts not focused on any particular client project.
- Geological Survey of Canada, Natural Resources Canada: Supported the initial development of the ArcSDE raster driver.
- OSGIS and the Geo-Information and ICT Department of the Ministry of Transport, Public Works and Water Management: Funded the DWG/DXF writing driver in OGR.
- Geosoft: Supported improvements to libtiff (RGBA Strip/Tile access), and the Arc/Info Binary Grid driver.
- Geospace Inc, Supported the development of write functionality for the OGR ArcSDE driver.
- GeoTango: Supported OGR Memory driver, Virtual Raster Filtering, and NITF RPC capabilities.
- i-cubed: Supported the MrSID driver.
- Intergraph: Supported development of the Erdas Imagine driver.
- Keyhole: Supported development of Erdas Imagine driver, and the GDAL Warp API.
- OPeNDAP: Supported development of the OGR OPeNDAP Driver.
- PCI Geomatics: Supported development of the JPEG2000 (JP2KAK) driver.
- Pixia: Supported NITF/JPEG2000 read support.
- UN FAO: Supported development of the IDA (WinDisp) driver, and GDAL VB6 bindings.
- SoftMap: Supported initial development of OGR as well as the OGR MapInfo integration.
- SRC: Supported development of the OGR OCI (Oracle Spatial) driver.
- Safe Software: Supported development of the OGR OLE DB provider, TIGER/Line driver, S-57 driver, DTED driver, FMEObjects driver, SDTS driver and NTF driver.
- Yukon Department of the Environment: Supported development of CDED / USGS DEM Writer.

Chapter 3

GDAL Downloads

This page has been moved to the wiki with a topic on downloading binaries (pre-built executables and a topic on downloading source.

Chapter 4

Simple C Example: gdalinfo.c

Chapter 5

Standard Driver Registration: gdalallregister.cpp

Chapter 6

Sample Driver: jdemdataset.cpp

Chapter 7

NEWS

Chapter 8

Building GDAL From Source

This topic is now lives in the wiki at: http://trac.osgeo.org/gdal/wiki/BuildHints

Chapter 9

GDAL Data Model

This document attempts to describe the GDAL data model. That is the types of information that a GDAL data store can contain, and their semantics.

9.1 Dataset

A dataset (represented by the GDALDataset class) is an assembly of related raster bands and some information common to them all. In particular the dataset has a concept of the raster size (in pixels and lines) that applies to all the bands. The dataset is also responsible for the georeferencing transform and coordinate system definition of all bands. The dataset itself can also have associated metadata, a list of name/value pairs in string form.

Note that the GDAL dataset, and raster band data model is loosely based on the OpenGIS Grid Coverages specification.

9.1.1 Coordinate System

Dataset coordinate systems are represented as OpenGIS Well Known Text strings. This can contain:

- An overall coordinate system name.
- A geographic coordinate system name.
- A datum identifier.
- An ellipsoid name, semi-major axis, and inverse flattening.
- A prime meridian name and offset from Greenwich.
- A projection method type (ie. Transverse Mercator).
- A list of projection parameters (ie. central_meridian).
- A units name, and conversion factor to meters or radians.
- Names and ordering for the axes.
- Codes for most of the above in terms of predefined coordinate systems from authorities such as EPSG.

For more information on OpenGIS WKT coordinate system definitions, and mechanisms to manipulate them, refer to the `osr_tutorial` document and/or the OGRSpatialReference class documentation.

The coordinate system returned by GDALDataset::GetProjectionRef() describes the georeferenced coordinates implied by the affine georeferencing transform returned by GDALDataset::GetGeoTransform(). The coordinate system returned by GDALDataset::GetGCPProjection() describes the georeferenced coordinates of the GCPs returned by GDALDataset::GetGCPs().

Note that a returned coordinate system strings of "" indicates nothing is known about the georeferencing coordinate system.

9.1.2 Affine GeoTransform

GDAL datasets have two ways of describing the relationship between raster positions (in pixel/line coordinates) and georeferenced coordinates. The first, and most commonly used is the affine transform (the other is GCPs).

The affine transform consists of six coefficients returned by GDALDataset::GetGeoTransform() which map pixel/line coordinates into georeferenced space using the following relationship:

```
Xgeo = GT(0) + Xpixel*GT(1) + Yline*GT(2)
Ygeo = GT(3) + Xpixel*GT(4) + Yline*GT(5)
```

In case of north up images, the GT(2) and GT(4) coefficients are zero, and the GT(1) is pixel width, and GT(5) is pixel height. The (GT(0),GT(3)) position is the top left corner of the top left pixel of the raster.

Note that the pixel/line coordinates in the above are from (0.0,0.0) at the top left corner of the top left pixel to (width_in_pixels,height_in_pixels) at the bottom right corner of the bottom right pixel. The pixel/line location of the center of the top left pixel would therefore be (0.5,0.5).

9.1.3 GCPs

A dataset can have a set of control points relating one or more positions on the raster to georeferenced coordinates. All GCPs share a georeferencing coordinate system (returned by GDALDataset::GetGCPProjection()). Each GCP (represented as the GDAL_GCP class) contains the following:

```
typedef struct
{
    char *pszId;
    char *pszInfo;
    double  dfGCPPixel;
    double dfGCPLine;
    double dfGCPX;
    double dfGCPY;
    double dfGCPZ;
} GDAL_GCP;
```

The pszId string is intended to be a unique (and often, but not always numerical) identifier for the GCP within the set of GCPs on this dataset. The pszInfo is usually an empty string, but can contain any user defined text associated with the GCP. Potentially this can also contain machine parsable information on GCP status though that isn't done at this time.

The (Pixel,Line) position is the GCP location on the raster. The (X,Y,Z) position is the associated georeferenced location with the Z often being zero.

The GDAL data model does not imply a transformation mechanism that must be generated from the GCPs ... this is left to the application. However 1st to 5th order polynomials are common.

Normally a dataset will contain either an affine geotransform, GCPs or neither. It is uncommon to have both, and it is undefined which is authoritative.

9.1.4 Metadata

GDAL metadata is auxiliary format and application specific textual data kept as a list of name/value pairs. The names are required to be well behaved tokens (no spaces, or odd characters). The values can be of any length, and contain anything except an embedded null (ASCII zero).

The metadata handling system is not well tuned to handling very large bodies of metadata. Handling of more than 100K of metadata for a dataset is likely to lead to performance degradation.

Some formats will support generic (user defined) metadata, while other format drivers will map specific format fields to metadata names. For instance the TIFF driver returns a few information tags as metadata including the date/time field which is returned as:

```
TIFFTAG_DATETIME=1999:05:11 11:29:56
```

Metadata is split into named groups called domains, with the default domain having no name (NULL or ""). Some specific domains exist for special purposes. Note that currently there is no way to enumerate all the domains available for a given object, but applications can "test" for any domains they know how to interprete.

The following metadata items have well defined semantics in the default domain:

- AREA_OR_POINT: May be either "Area" (the default) or "Point". Indicates whether a pixel value should be assumed to represent a sampling over the region of the pixel or a point sample at the center of the pixel. This is not intended to influence interpretation of georeferencing which remains area oriented.
- NODATA_VALUES: The value is a list of space separated pixel values matching the number of bands in the dataset that can be collectively used to identify pixels that are nodata in the dataset. With this style of nodata a pixel is considered nodata in all bands if and only if all bands match the corresponding value in the NODATA_VALUES tuple. This metadata is not widely honoured by GDAL drivers, algorithms or utilities at this time.
- MATRIX_REPRESENTATION: This value, used for Polarimetric SAR datasets, contains the matrix representation that this data is provided in. The following are acceptable values:
  - SCATTERING
  - SYMMETRIZED_SCATTERING
  - COVARIANCE
  - SYMMETRIZED_COVARIANCE
  - COHERENCY
  - SYMMETRIZED_COHERENCY
  - KENNAUGH
  - SYMMETRIZED_KENNAUGH
- POLARMETRIC_INTERP: This metadata item is defined for Raster Bands for polarimetric SAR data. This indicates which entry in the specified matrix representation of the data this band represents. For a dataset provided as a scattering matrix, for example, acceptable values for this metadata item are HH, HV, VH, VV. When the dataset is a covariance matrix, for example, this metadata item will be one of Covariance_11, Covariance_22, Covariance_33, Covariance_12, Covariance_13, Covariance_23 (since the matrix itself is a hermitian matrix, that is all the data that is required to describe the matrix).

9.1.4.1 SUBDATASETS Domain

The SUBDATASETS domain holds a list of child datasets. Normally this is used to provide pointers to a list of images stored within a single multi image file (such as HDF or NITF). For instance, an NITF with four images might have the following subdataset list.

```
SUBDATASET_1_NAME=NITF_IM:0:multi_1b.ntf
SUBDATASET_1_DESC=Image 1 of multi_1b.ntf
SUBDATASET_2_NAME=NITF_IM:1:multi_1b.ntf
SUBDATASET_2_DESC=Image 2 of multi_1b.ntf
SUBDATASET_3_NAME=NITF_IM:2:multi_1b.ntf
SUBDATASET_3_DESC=Image 3 of multi_1b.ntf
```

```
SUBDATASET_4_NAME=NITF_IM:3:multi_1b.ntf
SUBDATASET_4_DESC=Image 4 of multi_1b.ntf
SUBDATASET_5_NAME=NITF_IM:4:multi_1b.ntf
SUBDATASET_5_DESC=Image 5 of multi_1b.ntf
```

The value of the _NAME is the string that can be passed to GDALOpen() to access the file. The _DESC value is intended to be a more user friendly string that can be displayed to the user in a selector.

9.1.4.2 IMAGE_STRUCTURE Domain

Metadata in the default domain is intended to be related to the image, and not particularly related to the way the image is stored on disk. That is, it is suitable for copying with the dataset when it is copied to a new format. Some information of interest is closely tied to a particular file format and storage mechanism. In order to prevent this getting copied along with datasets it is placed in a special domain called IMAGE_-STRUCTURE that should not normally be copied to new formats.

Currently the following items are defined by RFC 14 as having specific semantics in the IMAGE_-STRUCTURE domain.

- COMPRESSION: The compression type used for this dataset or band. There is no fixed catalog of compression type names, but where a given format includes a COMPRESSION creation option, the same list of values should be used here as there.
- NBITS: The actual number of bits used for this band, or the bands of this dataset. Normally only present when the number of bits is non-standard for the datatype, such as when a 1 bit TIFF is represented through GDAL as GDT_Byte.
- INTERLEAVE: This only applies on datasets, and the value should be one of PIXEL, LINE or BAND. It can be used as a data access hint.
- PIXELTYPE: This may appear on a GDT_Byte band (or the corresponding dataset) and have the value SIGNEDBYTE to indicate the unsigned byte values between 128 and 255 should be interpreted as being values between -128 and -1 for applications that recognise the SIGNEDBYTE type.

9.1.4.3 RPC Domain

The RPC metadata domain holds metadata describing the Rational Polynomial Coefficient geometry model for the image if present. This geometry model can be used to transform between pixel/line and georeferenced locations. The items defining the model are:

- ERR_BIAS: Error - Bias. The RMS bias error in meters per horizontal axis of all points in the image (-1.0 if unknown)
- ERR_RAND: Error - Random. RMS random error in meters per horizontal axis of each point in the image (-1.0 if unknown)
- LINE_OFF: Line Offset
- SAMP_OFF: Sample Offset
- LAT_OFF: Geodetic Latitude Offset
- LONG_OFF: Geodetic Longitude Offset
- HEIGHT_OFF: Geodetic Height Offset

- LINE_SCALE: Line Scale
- SAMP_SCALE: Sample Scale
- LAT_SCALE: Geodetic Latitude Scale
- LONG_SCALE: Geodetic Longitude Scale
- HEIGHT_SCALE: Geodetic Height Scale
- LINE_NUM_COEFF (1-20): Line Numerator Coefficients. Twenty coefficients for the polynomial in the Numerator of the rn equation. (space separated)
- LINE_DEN_COEFF (1-20): Line Denominator Coefficients. Twenty coefficients for the polynomial in the Denominator of the rn equation. (space separated)
- SAMP_NUM_COEFF (1-20): Sample Numerator Coefficients. Twenty coefficients for the polynomial in the Numerator of the cn equation. (space separated)
- SAMP_DEN_COEFF (1-20): Sample Denominator Coefficients. Twenty coefficients for the polynomial in the Denominator of the cn equation. (space separated)

These fields are directly derived from the document prospective GeoTIFF RPC document (http://geotiff.maptools.org/rpc_prop.html) which in turn is closely modelled on the NITF RPC00B definition.

9.1.4.4 xml: Domains

Any domain name prefixed with "xml:" is not normal name/value metadata. It is a single XML document stored in one big string.

9.2 Raster Band

A raster band is represented in GDAL with the GDALRasterBand class. It represents a single raster band/channel/layer. It does not necessarily represent a whole image. For instance, a 24bit RGB image would normally be represented as a dataset with three bands, one for red, one for green and one for blue.

A raster band has the following properties:

- A width and height in pixels and lines. This is the same as that defined for the dataset, if this is a full resolution band.
- A datatype (GDALDataType). One of Byte, UInt16, Int16, UInt32, Int32, Float32, Float64, and the complex types CInt16, CInt32, CFloat32, and CFloat64.
- A block size. This is a preferred (efficient) access chunk size. For tiled images this will be one tile. For scanline oriented images this will normally be one scanline.
- A list of name/value pair metadata in the same format as the dataset, but of information that is potentially specific to this band.
- An optional description string.
- An optional single nodata pixel value (see also NODATA_VALUES metadata on the dataset for multi-band style nodata values).

- An optional nodata mask band marking pixels as nodata or in some cases transparency as discussed in RFC 15: Band Masks.
- An optional list of category names (effectively class names in a thematic image).
- An optional minimum and maximum value.
- An optional offset and scale for transforming raster values into meaning full values (ie translate height to meters)
- An optional raster unit name. For instance, this might indicate linear units for elevation data.
- A color interpretation for the band. This is one of:
  - GCI_Undefined: the default, nothing is known.
  - GCI_GrayIndex: this is an independent grayscale image
  - GCI_PaletteIndex: this raster acts as an index into a color table
  - GCI_RedBand: this raster is the red portion of an RGB or RGBA image
  - GCI_GreenBand: this raster is the green portion of an RGB or RGBA image
  - GCI_BlueBand: this raster is the blue portion of an RGB or RGBA image
  - GCI_AlphaBand: this raster is the alpha portion of an RGBA image
  - GCI_HueBand: this raster is the hue of an HLS image
  - GCI_SaturationBand: this raster is the saturation of an HLS image
  - GCI_LightnessBand: this raster is the hue of an HLS image
  - GCI_CyanBand: this band is the cyan portion of a CMY or CMYK image
  - GCI_MagentaBand: this band is the magenta portion of a CMY or CMYK image
  - GCI_YellowBand: this band is the yellow portion of a CMY or CMYK image
  - GCI_BlackBand: this band is the black portion of a CMYK image.
- A color table, described in more detail later.
- Knowledge of reduced resolution overviews (pyramids) if available.

9.3 Color Table

A color table consists of zero or more color entries described in C by the following structure:

```
typedef struct
{
    /- gray, red, cyan or hue -/
    short      c1;

    /- green, magenta, or lightness -/
    short      c2;

    /- blue, yellow, or saturation -/
    short      c3;

    /- alpha or blackband -/
    short      c4;
} GDALColorEntry;
```

The color table also has a palette interpretation value (GDALPaletteInterp) which is one of the following values, and indicates how the c1/c2/c3/c4 values of a color entry should be interpreted.

- GPI_Gray: Use c1 as grayscale value.
- GPI_RGB: Use c1 as red, c2 as green, c3 as blue and c4 as alpha.
- GPI_CMYK: Use c1 as cyan, c2 as magenta, c3 as yellow and c4 as black.
- GPI_HLS: Use c1 as hue, c2 as lightness, and c3 as saturation.

To associate a color with a raster pixel, the pixel value is used as a subscript into the color table. That means that the colors are always applied starting at zero and ascending. There is no provision for indicating a prescaling mechanism before looking up in the color table.

9.4 Overviews

A band may have zero or more overviews. Each overview is represented as a "free standing" GDALRasterBand. The size (in pixels and lines) of the overview will be different than the underlying raster, but the geographic region covered by overviews is the same as the full resolution band.

The overviews are used to display reduced resolution overviews more quickly than could be done by reading all the full resolution data and downsampling.

Bands also have a HasArbitraryOverviews property which is TRUE if the raster can be read at any resolution efficiently but with no distinct overview levels. This applies to some FFT encoded images, or images pulled through gateways (like OGDI) where downsampling can be done efficiently at the remote point.

Chapter 10

GDAL Driver Implementation Tutorial

10.1 Overall Approach

In general new formats are added to GDAL by implementing format specific drivers as subclasses of GDALDataset, and band accessors as subclasses of GDALRasterBand. As well, a GDALDriver instance is created for the format, and registered with the GDALDriverManager, to ensure that the system *knows* about the format.

This tutorial will start with implementing a simple read-only driver (based on the JDEM driver), and then proceed to utilizing the RawRasterBand helper class, implementing creatable and updatable formats, and some esoteric issues.

It is strongly advised that the `GDAL Data Model` description be reviewed and understood before attempting to implement a GDAL driver.

10.2 Contents

1. Implementing the Dataset
2. Implementing the RasterBand
3. The Driver
4. Adding Driver to GDAL Tree
5. Adding Georeferencing
6. Overviews
7. File Creation
8. RawDataset/RawRasterBand Helper Classes
9. Metadata, and Other Exotic Extensions

10.3 Implementing the Dataset

We will start showing minimal implementation of a read-only driver for the Japanese DEM format (`jdemdataset.cpp`). First we declare a format specific dataset class, JDEMDataset in this case.

```
class JDEMDataset : public GDALDataset
{
    FILE        *fp;
    GByte       abyHeader[1012];

  public:
                ~JDEMDataset();

    static GDALDataset *Open( GDALOpenInfo * );
};
```

In general we provide capabilities for a driver, by overriding the various virtual methods on the GDALDataset base class. However, the Open() method is special. This is not a virtual method on the base class, and we will need a freestanding function for this operation, so we declare it static. Implementing it as a method in the JDEMDataset class is convenient because we have privileged access to modify the contents of the database object.

The open method itself may look something like this:

```
GDALDataset *JDEMDataset::Open( GDALOpenInfo * poOpenInfo )

{
// ----------------------------------------------------------------------
//      Before trying JDEMOpen() we first verify that there is at
//      least one "\n#keyword" type signature in the first chunk of
//      the file.
// ----------------------------------------------------------------------
    if( poOpenInfo->fp == NULL || poOpenInfo->nHeaderBytes < 50 )
        return NULL;

    // check if century values seem reasonable
    if( (!EQUALN((char *)poOpenInfo->pabyHeader+11,"19",2)
          && !EQUALN((char *)poOpenInfo->pabyHeader+11,"20",2))
        || (!EQUALN((char *)poOpenInfo->pabyHeader+15,"19",2)
            && !EQUALN((char *)poOpenInfo->pabyHeader+15,"20",2))
        || (!EQUALN((char *)poOpenInfo->pabyHeader+19,"19",2)
            && !EQUALN((char *)poOpenInfo->pabyHeader+19,"20",2)) )
    {
        return NULL;
    }

// ----------------------------------------------------------------------
//      Create a corresponding GDALDataset.
// ----------------------------------------------------------------------
    JDEMDataset  *poDS;

    poDS = new JDEMDataset();

    poDS->fp = poOpenInfo->fp;
    poOpenInfo->fp = NULL;

// ----------------------------------------------------------------------
//      Read the header.
// ----------------------------------------------------------------------
    VSIFSeek( poDS->fp, 0, SEEK_SET );
    VSIFRead( poDS->abyHeader, 1, 1012, poDS->fp );

    poDS->nRasterXSize = JDEMGetField( (char *) poDS->abyHeader + 23, 3 );
    poDS->nRasterYSize = JDEMGetField( (char *) poDS->abyHeader + 26, 3 );

// ----------------------------------------------------------------------
//      Create band information objects.
// ----------------------------------------------------------------------
    poDS->nBands = 1;
    poDS->SetBand( 1, new JDEMRasterBand( poDS, 1 ));

    return( poDS );
}
```

The first step in any database Open function is to verify that the file being passed is in fact of the type this driver is for. It is important to realize that each driver's Open function is called in turn till one succeeds. Drivers must quietly return NULL if the passed file is not of their format. They should only produce an error if the file does appear to be of their supported format, but is for some reason unsupported or corrupt.

The information on the file to be opened is passed in contained in a GDALOpenInfo object. The GDALOpenInfo includes the following public data members:

```
    char        *pszFilename;

    GDALAccess  eAccess; // GA_ReadOnly or GA_Update

    GBool       bStatOK;
    VSIStatBuf  sStat;

    FILE        *fp;
```

```
int         nHeaderBytes;
GByte       *pabyHeader;
```

The driver can inspect these to establish if the file is supported. If the pszFilename refers to an object in the file system, the **bStatOK** flag will be set, and the **sStat** structure will contain normal stat() information about the object (be it directory, file, device). If the object is a regular readable file, the **fp** will be non-NULL, and can be used for reads on the file (please use the VSI stdio functions from cpl_vsi.h). As well, if the file was successfully opened, the first kilobyte or so is read in, and put in **pabyHeader**, with the exact size in **nHeaderBytes**.

In this typical testing example it is verified that the file was successfully opened, that we have at least enough header information to perform our test, and that various parts of the header are as expected for this format. In this case, there are no *magic* numbers for JDEM format so we check various date fields to ensure they have reasonable century values. If the test fails, we quietly return NULL indicating this file isn't of our supported format.

```
if( poOpenInfo->fp == NULL || poOpenInfo->nHeaderBytes < 50 )
    return NULL;

// check if century values seem reasonable
if( (!EQUALN((char *)poOpenInfo->pabyHeader+11,"19",2)
      && !EQUALN((char *)poOpenInfo->pabyHeader+11,"20",2))
    || (!EQUALN((char *)poOpenInfo->pabyHeader+15,"19",2)
         && !EQUALN((char *)poOpenInfo->pabyHeader+15,"20",2))
    || (!EQUALN((char *)poOpenInfo->pabyHeader+19,"19",2)
         && !EQUALN((char *)poOpenInfo->pabyHeader+19,"20",2)) )
{
    return NULL;
}
```

It is important to make the *is this my format* test as stringent as possible. In this particular case the test is weak, and a file that happened to have 19s or 20s at a few locations could be erroneously recognized as JDEM format, causing it to not be handled properly.

Once we are satisfied that the file is of our format, we need to create an instance of the database class in which we will set various information of interest.

```
JDEMDataset         *poDS;

poDS = new JDEMDataset();

poDS->fp = poOpenInfo->fp;
poOpenInfo->fp = NULL;
```

Generally at this point we would open the file, to acquire a file handle for the dataset; however, if read-only access is sufficient it is permitted to **assume ownership** of the FILE * from the GDALOpenInfo object. Just ensure that it is set to NULL in the GDALOpenInfo to avoid having it get closed twice. It is also important to note that the state of the FILE * adopted is indeterminate. Ensure that the current location is reset with VSIFSeek() before assuming you can read from it. This is accomplished in the following statements which reset the file and read the header.

```
VSIFSeek( poDS->fp, 0, SEEK_SET );
VSIFRead( poDS->abyHeader, 1, 1012, poDS->fp );
```

Next the X and Y size are extracted from the header. The nRasterXSize and nRasterYSize are data fields inherited from the GDALDataset base class, and must be set by the Open() method.

```
    poDS->nRasterXSize = JDEMGetField( (char *) poDS->abyHeader + 23, 3 );
    poDS->nRasterYSize = JDEMGetField( (char *) poDS->abyHeader + 26, 3 );
```

Finally, all the bands related to this dataset must be attached using the SetBand() method. We will explore the JDEMRasterBand() class shortly.

```
    poDS->SetBand( 1, new JDEMRasterBand( poDS, 1 ));

    return( poDS );
```

10.4 Implementing the RasterBand

Similar to the customized JDEMDataset class subclassed from GDALDataset, we also need to declare and implement a customized JDEMRasterBand derived from GDALRasterBand for access to the band(s) of the JDEM file. For JDEMRasterBand the declaration looks like this:

```
class JDEMRasterBand : public GDALRasterBand
{
  public:
                JDEMRasterBand( JDEMDataset *, int );
    virtual CPLErr IReadBlock( int, int, void * );
};
```

The constructor may have any signature, and is only called from the Open() method. Other virtual methods, such as IReadBlock() must be exactly matched to the method signature in gdal_priv.h.

The constructor implementation looks like this:

```
JDEMRasterBand::JDEMRasterBand( JDEMDataset *poDS, int nBand )

{
    this->poDS = poDS;
    this->nBand = nBand;

    eDataType = GDT_Float32;

    nBlockXSize = poDS->GetRasterXSize();
    nBlockYSize = 1;
}
```

The following data members are inherited from GDALRasterBand, and should generally be set in the band constructor.

- **poDS**: Pointer to the parent GDALDataset.
- **nBand**: The band number within the dataset.
- **eDataType**: The data type of pixels in this band.
- **nBlockXSize**: The width of one block in this band.
- **nBlockYSize**: The height of one block in this band.

The full set of possible GDALDataType values are declared in gdal.h, and include GDT_Byte, GDT_UInt16, GDT_Int16, and GDT_Float32. The block size is used to establish a *natural* or efficient block size to access the data with. For tiled datasets this will be the size of a tile, while for most other datasets it will be one scanline, as in this case.

Next we see the implementation of the code that actually reads the image data, IReadBlock().

```
CPLErr JDEMRasterBand::IReadBlock( int nBlockXOff, int nBlockYOff,
                                   void * pImage )

{
    JDEMDataset *poGDS = (JDEMDataset *) poDS;
    char        *pszRecord;
    int         nRecordSize = nBlockXSize*5 + 9 + 2;
    int         i;

    VSIFSeek( poGDS->fp, 1011 + nRecordSize*nBlockYOff, SEEK_SET );

    pszRecord = (char *) CPLMalloc(nRecordSize);
    VSIFRead( pszRecord, 1, nRecordSize, poGDS->fp );

    if( !EQUALN((char *) poGDS->abyHeader,pszRecord,6) )
    {
        CPLFree( pszRecord );

        CPLError( CE_Failure, CPLE_AppDefined,
                  "JDEM Scanline corrupt.  Perhaps file was not transferred\n"
                  "in binary mode?" );
        return CE_Failure;
    }

    if( JDEMGetField( pszRecord + 6, 3 ) != nBlockYOff + 1 )
    {
        CPLFree( pszRecord );

        CPLError( CE_Failure, CPLE_AppDefined,
                  "JDEM scanline out of order, JDEM driver does not\n"
                  "currently support partial datasets." );
        return CE_Failure;
    }

    for( i = 0; i < nBlockXSize; i++ )
        ((float *) pImage)[i] = JDEMGetField( pszRecord + 9 + 5 * i, 5) * 0.1;

    return CE_None;
}
```

Key items to note are:

- It is typical to cast the GDALRasterBand::poDS member to the derived type of the owning dataset. If your RasterBand class will need privileged access to the owning dataset object, ensure it is declared as a friend (omitted above for brevity).
- If an error occurs, report it with CPLError(), and return CE_Failure. Otherwise return CE_None.
- The pImage buffer should be filled with one block of data. The block is the size declared in nBlockXSize and nBlockYSize for the raster band. The type of the data within pImage should match the type declared in eDataType in the raster band object.
- The nBlockXOff and nBlockYOff are block offsets, so with 128x128 tiled datasets values of 1 and 1 would indicate the block going from (128,128) to (255,255) should be loaded.

10.5 The Driver

While the JDEMDataset and JDEMRasterBand are now ready to use to read image data, it still isn't clear how the GDAL system knows about the new driver. This is accomplished via the GDALDriverManager. To register our format we implement a registration function:

```
CPL_C_START
void    GDALRegister_JDEM(void);
CPL_C_END

...

void GDALRegister_JDEM()

{
    GDALDriver  *poDriver;

    if( GDALGetDriverByName( "JDEM" ) == NULL )
    {
        poDriver = new GDALDriver();

        poDriver->SetDescription( "JDEM" );
        poDriver->SetMetadataItem( GDAL_DMD_LONGNAME,
                                   "Japanese DEM (.mem)" );
        poDriver->SetMetadataItem( GDAL_DMD_HELPTOPIC,
                                   "frmt_various.html#JDEM" );
        poDriver->SetMetadataItem( GDAL_DMD_EXTENSION, "mem" );

        poDriver->pfnOpen = JDEMDataset::Open;

        GetGDALDriverManager()->RegisterDriver( poDriver );
    }
}
```

The registration function will create an instance of a GDALDriver object when first called, and register it with the GDALDriverManager. The following fields can be set in the driver before registering it with the GDALDriverManager().

- The description is the short name for the format. This is a unique name for this format, often used to identity the driver in scripts and commandline programs. Normally 3-5 characters in length, and matching the prefix of the format classes. (mandatory)
- GDAL_DMD_LONGNAME: A longer descriptive name for the file format, but still no longer than 50-60 characters. (mandatory)
- GDAL_DMD_HELPTOPIC: The name of a help topic to display for this driver, if any. In this case JDEM format is contained within the various format web page held in gdal/html. (optional)
- GDAL_DMD_EXTENSION: The extension used for files of this type. If more than one pick the primary extension, or none at all. (optional)
- GDAL_DMD_MIMETYPE: The standard mime type for this file format, such as "image/png". (optional)
- GDAL_DMD_CREATIONOPTIONLIST: There is evolving work on mechanisms to describe creation options. See the geotiff driver for an example of this. (optional)
- GDAL_DMD_CREATIONDATATYPES: A list of space separated data types supported by this create when creating new datasets. If a Create() method exists, these will be will supported. If a CreateCopy() method exists, this will be a list of types that can be losslessly exported but it may include weaker data types than the type eventually written. For instance, a format with a CreateCopy() method, and that always writes Float32 might also list Byte, Int16, and UInt16 since they can losslessly translated to Float32. An example value might be "Byte Int16 UInt16". (required - if creation supported)
- pfnOpen: The function to call to try opening files of this format. (optional)
- pfnCreate: The function to call to create new updatable datasets of this format. (optional)

- pfnCreateCopy: The function to call to create a new dataset of this format copied from another source, but not necessary updatable. (optional)
- pfnDelete: The function to call to delete a dataset of this format. (optional)
- pfnUnloadDriver: A function called only when the driver is destroyed. Could be used to cleanup data at the driver level. Rarely used. (optional)

10.6 Adding Driver to GDAL Tree

Note that the GDALRegister_JDEM() method must be called by the higher level program in order to have access to the JDEM driver. Normal practice when writing new drivers is to:

1. Add a driver directory under gdal/frmts, with the directory name the same as the short name.
2. Add a GNUmakefile and makefile.vc in that directory modelled on those from other similar directories (ie. the jdem directory).
3. Add the module with the dataset, and rasterband implementation. Generally this is called <short_name>dataset.cpp, with all the GDAL specific code in one file, though that is not required.
4. Add the registration entry point declaration (ie. GDALRegister_JDEM()) to gdal/gcore/gdal_frmts.h.
5. Add a call to the registration function to frmts/gdalallregister.c, protected by an appropriate ifdef.
6. Add the format short name to the GDAL_FORMATS macro in GDALmake.opt.in (and to GDALmake.opt).
7. Add a format specific item to the EXTRAFLAGS macro in frmts/makefile.vc.

Once this is all done, it should be possible to rebuild GDAL, and have the new format available in all the utilities. The gdalinfo utility can be used to test that opening and reporting on the format is working, and the gdal_translate utility can be used to test image reading.

10.7 Adding Georeferencing

Now we will take the example a step forward, adding georeferencing support. We add the following two virtual method overrides to JDEMDataset, taking care to exactly match the signature of the method on the GDALRasterDataset base class.

```
    CPLErr      GetGeoTransform( double * padfTransform );
    const char *GetProjectionRef();
```

The implementation of GetGeoTransform() just copies the usual geotransform matrix into the supplied buffer. Note that GetGeoTransform() may be called a lot, so it isn't generally wise to do a lot of computation in it. In many cases the Open() will collect the geotransform, and this method will just copy it over. Also note that the geotransform return is based on an anchor point at the top left corner of the top left pixel, not the center of pixel approach used in some packages.

```
CPLErr JDEMDataset::GetGeoTransform( double * padfTransform )

{
    double      dfLLLat, dfLLLong, dfURLat, dfURLong;
```

```
    dfLLLat = JDEMGetAngle( (char *) abyHeader + 29 );
    dfLLLong = JDEMGetAngle( (char *) abyHeader + 36 );
    dfURLat = JDEMGetAngle( (char *) abyHeader + 43 );
    dfURLong = JDEMGetAngle( (char *) abyHeader + 50 );

    padfTransform[0] = dfLLLong;
    padfTransform[3] = dfURLat;
    padfTransform[1] = (dfURLong - dfLLLong) / GetRasterXSize();
    padfTransform[2] = 0.0;

    padfTransform[4] = 0.0;
    padfTransform[5] = -1 * (dfURLat - dfLLLat) / GetRasterYSize();


    return CE_None;
}
```

The GetProjectionRef() method returns a pointer to an internal string containing a coordinate system definition in OGC WKT format. In this case the coordinate system is fixed for all files of this format, but in more complex cases a definition may need to be composed on the fly, in which case it may be helpful to use the OGRSpatialReference class to help build the definition.

```
const char *JDEMDataset::GetProjectionRef()

{
    return( "GEOGCS[\"Tokyo\",DATUM[\"Tokyo\",SPHEROID[\"Bessel 1841\","
        "6377397.155,299.1528128,AUTHORITY[\"EPSG\",7004]],TOWGS84[-148,"
        "507,685,0,0,0,0],AUTHORITY[\"EPSG\",6301]],PRIMEM[\"Greenwich\","
        "0,AUTHORITY[\"EPSG\",8901]],UNIT[\"DMSH\",0.0174532925199433,"
        "AUTHORITY[\"EPSG\",9108]],AXIS[\"Lat\",NORTH],AXIS[\"Long\",EAST],"
        "AUTHORITY[\"EPSG\",4301]]" );
}
```

This completes explanation of the features of the JDEM driver. The full source for jdemdataset.cpp can be reviewed as needed.

10.8 Overviews

GDAL allows file formats to make pre-built overviews available to applications via the GDALRasterBand::GetOverview() and related methods. However, implementing this is pretty involved, and goes beyond the scope of this document for now. The GeoTIFF driver (gdal/frmts/gtiff/geotiff.cpp) and related source can be reviewed for an example of a file format implementing overview reporting and creation support.

Formats can also report that they have arbitrary overviews, by overriding the HasArbitraryOverviews() method on the GDALRasterBand, returning TRUE. In this case the raster band object is expected to override the RasterIO() method itself, to implement efficient access to imagery with resampling. This is also involved, and there are a lot of requirements for correct implementation of the RasterIO() method. An example of this can be found in the OGDI and ECW formats.

However, by far the most common approach to implementing overviews is to use the default support in GDAL for external overviews stored in TIFF files with the same name as the dataset, but the extension .ovr appended. In order to enable reading and creation of this style of overviews it is necessary for the GDALDataset to initialize the oOvManager object within itself. This is typically accomplished with a call like the following near the end of the Open() method.

```
    poDS->oOvManager.Initialize( poDS, poOpenInfo->pszFilename );
```

This will enable default implementations for reading and creating overviews for the format. It is advised that this be enabled for all simple file system based formats unless there is a custom overview mechanism to be tied into.

10.9 File Creation

There are two approaches to file creation. The first method is called the CreateCopy() method, and involves implementing a function that can write a file in the output format, pulling all imagery and other information needed from a source GDALDataset. The second method, the dynamic creation method, involves implementing a Create method to create the shell of the file, and then the application writes various information by calls to set methods.

The benefits of the first method are that that all the information is available at the point the output file is being created. This can be especially important when implementing file formats using external libraries which require information like colormaps, and georeferencing information at the point the file is created. The other advantage of this method is that the CreateCopy() method can read some kinds of information, such as min/max, scaling, description and GCPs for which there are no equivalent set methods.

The benefits of the second method are that applications can create an empty new file, and write results to it as they become available. A complete image of the desired data does not have to be available in advance.

For very important formats both methods may be implemented, otherwise do whichever is simpler, or provides the required capabilities.

10.9.1 CreateCopy

The GDALDriver::CreateCopy() method call is passed through directly, so that method should be consulted for details of arguments. However, some things to keep in mind are:

- If the bStrict flag is FALSE the driver should try to do something reasonable when it cannot exactly represent the source dataset, transforming data types on the fly, dropping georeferencing and so forth.
- Implementing progress reporting correctly is somewhat involved. The return result of the progress function needs always to be checked for cancellation, and progress should be reported at reasonable intervals. The JPEGCreateCopy() method demonstrates good handling of the progress function.
- Special creation options should be documented in the online help. If the options take the format "NAME=VALUE" the papszOptions list can be manipulated with CPLFetchNameValue() as demonstrated in the handling of the QUALITY and PROGRESSIVE flags for JPEGCreateCopy().
- The returned GDALDataset handle can be in ReadOnly or Update mode. Return it in Update mode if practical, otherwise in ReadOnly mode is fine.

The full implementation of the CreateCopy function for JPEG (which is assigned to pfnCreateCopy in the GDALDriver object) is here.

```
static GDALDataset *
JPEGCreateCopy( const char * pszFilename, GDALDataset *poSrcDS,
                int bStrict, char ** papszOptions,
                GDALProgressFunc pfnProgress, void * pProgressData )

{
    int  nBands = poSrcDS->GetRasterCount();
    int  nXSize = poSrcDS->GetRasterXSize();
    int  nYSize = poSrcDS->GetRasterYSize();
```

```
    int  nQuality = 75;
    int  bProgressive = FALSE;

// ----------------------------------------------------------------------
//      Some some rudimentary checks
// ----------------------------------------------------------------------
    if( nBands != 1 && nBands != 3 )
    {
        CPLError( CE_Failure, CPLE_NotSupported,
                  "JPEG driver doesn't support %d bands.  Must be 1 (grey) "
                  "or 3 (RGB) bands.\n", nBands );

        return NULL;
    }

    if( poSrcDS->GetRasterBand(1)->GetRasterDataType() != GDT_Byte && bStrict )
    {
        CPLError( CE_Failure, CPLE_NotSupported,
                  "JPEG driver doesn't support data type %s. "
                  "Only eight bit byte bands supported.\n",
                  GDALGetDataTypeName(
                      poSrcDS->GetRasterBand(1)->GetRasterDataType()) );

        return NULL;
    }

// ----------------------------------------------------------------------
//      What options has the user selected?
// ----------------------------------------------------------------------
    if( CSLFetchNameValue(papszOptions,"QUALITY") != NULL )
    {
        nQuality = atoi(CSLFetchNameValue(papszOptions,"QUALITY"));
        if( nQuality < 10 || nQuality > 100 )
        {
            CPLError( CE_Failure, CPLE_IllegalArg,
                      "QUALITY=%s is not a legal value in the range 10-100.",
                      CSLFetchNameValue(papszOptions,"QUALITY") );
            return NULL;
        }
    }

    if( CSLFetchNameValue(papszOptions,"PROGRESSIVE") != NULL )
    {
        bProgressive = TRUE;
    }

// ----------------------------------------------------------------------
//      Create the dataset.
// ----------------------------------------------------------------------
    FILE *fpImage;

    fpImage = VSIFOpen( pszFilename, "wb" );
    if( fpImage == NULL )
    {
        CPLError( CE_Failure, CPLE_OpenFailed,
                  "Unable to create jpeg file %s.\n",
                  pszFilename );
        return NULL;
    }

// ----------------------------------------------------------------------
//      Initialize JPG access to the file.
// ----------------------------------------------------------------------
    struct jpeg_compress_struct sCInfo;
    struct jpeg_error_mgr sJErr;

    sCInfo.err = jpeg_std_error( &sJErr );
```

```
    jpeg_create_compress( &sCInfo );

    jpeg_stdio_dest( &sCInfo, fpImage );

    sCInfo.image_width = nXSize;
    sCInfo.image_height = nYSize;
    sCInfo.input_components = nBands;

    if( nBands == 1 )
    {
        sCInfo.in_color_space = JCS_GRAYSCALE;
    }
    else
    {
        sCInfo.in_color_space = JCS_RGB;
    }

    jpeg_set_defaults( &sCInfo );

    jpeg_set_quality( &sCInfo, nQuality, TRUE );

    if( bProgressive )
        jpeg_simple_progression( &sCInfo );

    jpeg_start_compress( &sCInfo, TRUE );

// --------------------------------------------------------------------
//      Loop over image, copying image data.
// --------------------------------------------------------------------
    GByte  *pabyScanline;
    CPLErr      eErr;

    pabyScanline = (GByte *) CPLMalloc( nBands * nXSize );

    for( int iLine = 0; iLine < nYSize; iLine++ )
    {
        JSAMPLE      *ppSamples;

        for( int iBand = 0; iBand < nBands; iBand++ )
        {
            GDALRasterBand * poBand = poSrcDS->GetRasterBand( iBand+1 );
            eErr = poBand->RasterIO( GF_Read, 0, iLine, nXSize, 1,
                                     pabyScanline + iBand, nXSize, 1, GDT_Byte,
                                     nBands, nBands * nXSize );
        }

        ppSamples = pabyScanline;
        jpeg_write_scanlines( &sCInfo, &ppSamples, 1 );
    }

    CPLFree( pabyScanline );

    jpeg_finish_compress( &sCInfo );
    jpeg_destroy_compress( &sCInfo );

    VSIFClose( fpImage );

    return (GDALDataset *) GDALOpen( pszFilename, GA_ReadOnly );
}
```

10.9.2 Dynamic Creation

In the case of dynamic creation, there is no source dataset. Instead the size, number of bands, and pixel data type of the desired file is provided but other information (such as georeferencing, and imagery data) would be supplied later via other method calls on the resulting GDALDataset.

The following sample implement PCI .aux labelled raw raster creation. It follows a common approach of creating a blank, but valid file using non-GDAL calls, and then calling GDALOpen(,GA_Update) at the end to return a writable file handle. This avoids having to duplicate the various setup actions in the Open() function.

```
GDALDataset *PAuxDataset::Create( const char * pszFilename,
                                  int nXSize, int nYSize, int nBands,
                                  GDALDataType eType,
                                  char ** // papszParmList  )

{
    char *pszAuxFilename;

// --------------------------------------------------------------------
//      Verify input options.
// --------------------------------------------------------------------
    if( eType != GDT_Byte && eType != GDT_Float32 && eType != GDT_UInt16
        && eType != GDT_Int16 )
    {
        CPLError( CE_Failure, CPLE_AppDefined,
              "Attempt to create PCI .Aux labelled dataset with an illegal\n"
              "data type (%s).\n",
              GDALGetDataTypeName(eType) );

        return NULL;
    }

// --------------------------------------------------------------------
//      Try to create the file.
// --------------------------------------------------------------------
    FILE *fp;

    fp = VSIFOpen( pszFilename, "w" );

    if( fp == NULL )
    {
        CPLError( CE_Failure, CPLE_OpenFailed,
                  "Attempt to create file `%s' failed.\n",
                  pszFilename );
        return NULL;
    }

// --------------------------------------------------------------------
//      Just write out a couple of bytes to establish the binary
//      file, and then close it.
// --------------------------------------------------------------------
    VSIFWrite( (void *) "\0\0", 2, 1, fp );
    VSIFClose( fp );

// --------------------------------------------------------------------
//      Create the aux filename.
// --------------------------------------------------------------------
    pszAuxFilename = (char *) CPLMalloc(strlen(pszFilename)+5);
    strcpy( pszAuxFilename, pszFilename );;

    for( int i = strlen(pszAuxFilename)-1; i > 0; i-- )
    {
        if( pszAuxFilename[i] == '.' )
        {
            pszAuxFilename[i] = '\0';
            break;
        }
    }

    strcat( pszAuxFilename, ".aux" );

// --------------------------------------------------------------------
```

```
//      Open the file.
// ----------------------------------------------------------------------
    fp = VSIFOpen( pszAuxFilename, "wt" );
    if( fp == NULL )
    {
        CPLError( CE_Failure, CPLE_OpenFailed,
                  "Attempt to create file '%s' failed.\n",
                  pszAuxFilename );
        return NULL;
    }

// ----------------------------------------------------------------------
//      We need to write out the original filename but without any
//      path components in the AuxilaryTarget line.  Do so now.
// ----------------------------------------------------------------------
    int iStart;

    iStart = strlen(pszFilename)-1;
    while( iStart > 0 && pszFilename[iStart-1] != '/'
           && pszFilename[iStart-1] != '\\' )
        iStart--;

    VSIFPrintf( fp, "AuxilaryTarget: %s\n", pszFilename + iStart );

// ----------------------------------------------------------------------
//      Write out the raw definition for the dataset as a whole.
// ----------------------------------------------------------------------
    VSIFPrintf( fp, "RawDefinition: %d %d %d\n",
                nXSize, nYSize, nBands );

// ----------------------------------------------------------------------
//      Write out a definition for each band.  We always write band
//      sequential files for now as these are pretty efficiently
//      handled by GDAL.
// ----------------------------------------------------------------------
    int nImgOffset = 0;

    for( int iBand = 0; iBand < nBands; iBand++ )
    {
        const char * pszTypeName;
        int      nPixelOffset;
        int      nLineOffset;

        nPixelOffset = GDALGetDataTypeSize(eType)/8;
        nLineOffset = nXSize * nPixelOffset;

        if( eType == GDT_Float32 )
            pszTypeName = "32R";
        else if( eType == GDT_Int16 )
            pszTypeName = "16S";
        else if( eType == GDT_UInt16 )
            pszTypeName = "16U";
        else
            pszTypeName = "8U";

        VSIFPrintf( fp, "ChanDefinition-%d: %s %d %d %d %s\n",
                    iBand+1, pszTypeName,
                    nImgOffset, nPixelOffset, nLineOffset,
#ifdef CPL_LSB
                    "Swapped"
#else
                    "Unswapped"
#endif
                    );

        nImgOffset += nYSize * nLineOffset;
    }
```

```
// ----------------------------------------------------------------------
//      Cleanup
// ----------------------------------------------------------------------
    VSIFClose( fp );

    return (GDALDataset *) GDALOpen( pszFilename, GA_Update );
}
```

File formats supporting dynamic creation, or even just update-in-place access also need to implement an IWriteBlock() method on the raster band class. It has semantics similar to IReadBlock(). As well, for various esoteric reasons, it is critical that a FlushCache() method be implemented in the raster band destructor. This is to ensure that any write cache blocks for the band be flushed out before the destructor is called.

10.10 RawDataset/RawRasterBand Helper Classes

Many file formats have the actual imagery data stored in a regular, binary, scanline oriented format. Rather than re-implement the access semantics for this for each formats, there are provided RawDataset and RawRasterBand classes declared in gdal/frmts/raw that can be utilized to implement efficient and convenient access.

In these cases the format specific band class may not be required, or if required it can be derived from RawRasterBand. The dataset class should be derived from RawDataset.

The Open() method for the dataset then instantiates raster bands passing all the layout information to the constructor. For instance, the PNM driver uses the following calls to create it's raster bands.

```
    if( poOpenInfo->pabyHeader[1] == '5' )
    {
        poDS->SetBand(
            1, new RawRasterBand( poDS, 1, poDS->fpImage,
                                  iIn, 1, nWidth, GDT_Byte, TRUE ));
    }
    else
    {
        poDS->SetBand(
            1, new RawRasterBand( poDS, 1, poDS->fpImage,
                                  iIn, 3, nWidth*3, GDT_Byte, TRUE ));
        poDS->SetBand(
            2, new RawRasterBand( poDS, 2, poDS->fpImage,
                                  iIn+1, 3, nWidth*3, GDT_Byte, TRUE ));
        poDS->SetBand(
            3, new RawRasterBand( poDS, 3, poDS->fpImage,
                                  iIn+2, 3, nWidth*3, GDT_Byte, TRUE ));
    }
```

The RawRasterBand takes the following arguments.

- **poDS**: The GDALDataset this band will be a child of. This dataset must be of a class derived from RawRasterDataset.
- **nBand**: The band it is on that dataset, 1 based.
- **fpRaw**: The FILE * handle to the file containing the raster data.
- **nImgOffset**: The byte offset to the first pixel of raster data for the first scanline.
- **nPixelOffset**: The byte offset from the start of one pixel to the start of the next within the scanline.

- **nLineOffset**: The byte offset from the start of one scanline to the start of the next.
- **eDataType**: The GDALDataType code for the type of the data on disk.
- **bNativeOrder**: FALSE if the data is not in the same endianness as the machine GDAL is running on. The data will be automatically byte swapped.

Simple file formats utilizing the Raw services are normally placed all within one file in the gdal/frmts/raw directory. There are numerous examples there of format implementation.

10.11 Metadata, and Other Exotic Extensions

There are various other items in the GDAL data model, for which virtual methods exist on the GDALDataset and GDALRasterBand. They include:

- **Metadata**: Name/value text values about a dataset or band. The GDALMajorObject (base class for GDALRasterBand and GDALDataset) has built-in support for holding metadata, so for read access it only needs to be set with calls to SetMetadataItem() during the Open(). The SAR_CEOS (frmts/ceos2/sar_ceosdataset.cpp) and GeoTIFF drivers are examples of drivers implementing readable metadata.
- **ColorTables**: GDT_Byte raster bands can have color tables associated with them. The frmts/png/pngdataset.cpp driver contains an example of a format that supports colortables.
- **ColorInterpretation**: The PNG driver contains an example of a driver that returns an indication of whether a band should be treated as a Red, Green, Blue, Alpha or Greyscale band.
- **GCPs**: GDALDatasets can have a set of ground control points associated with them (as opposed to an explicit affine transform returned by GetGeotransform()) relating the raster to georeferenced coordinates. The MFF2 (gdal/frmts/raw/hkvdataset.cpp) format is a simple example of a format supporting GCPs.
- **NoDataValue**: Bands with known "nodata" values can implement the GetNoDataValue() method. See the PAux (frmts/raw/pauxdataset.cpp) for an example of this.
- **Category Names**: Classified images with names for each class can return them using the GetCategoryNames() method though no formats currently implement this.

Chapter 11

gdal_polygonize

produces a polygon feature layer from a raster

11.1 SYNOPSIS

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

11.2 DESCRIPTION

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

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

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

-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.

Chapter 12

gdal_proximity

produces a raster proximity map

12.1 SYNOPSIS

```
gdal_proximity.py srcfile dstfile [-srcband n] [-dstband n]
                  [-of format] [-co name=value]*
                  [-ot Byte/Int16/Int32/Float32/etc]
                  [-values n,n,n] [-distunits PIXEL/GEO]
                  [-maxdist n] [-nodata n] [-fixed-buf-val n]
```

12.2 DESCRIPTION

The gdal_proximity.py script generates a raster proximity map indicating the distance from the center of each pixel to the center of the nearest pixel identified as a target pixel. Target pixels are those in the source raster for which the raster pixel value is in the set of target pixel values.

srcfile The source raster file used to identify target pixels.

dstfile The destination raster file to which the proximity map will be written.

-srcband *n* Identifies the band in the source file to use (default is 1).

-srcband *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.

Chapter 13

GDAL API Tutorial

13.1 Opening the File

Before opening a GDAL supported raster datastore it is necessary to register drivers. There is a driver for each supported format. Normally this is accomplished with the GDALAllRegister() function which attempts to register all known drivers, including those auto-loaded from .so files using GDALDriverManager::AutoLoadDrivers(). If for some applications it is necessary to limit the set of drivers it may be helpful to review the code from gdalallregister.cpp.

Once the drivers are registered, the application should call the free standing GDALOpen() function to open a dataset, passing the name of the dataset and the access desired (GA_ReadOnly or GA_Update).

In C++:

```
#include "gdal_priv.h"

int main()
{
    GDALDataset  *poDataset;

    GDALAllRegister();

    poDataset = (GDALDataset *) GDALOpen( pszFilename, GA_ReadOnly );
    if( poDataset == NULL )
    {
        ...;
    }
```

In C:

```
#include "gdal.h"

int main()
{
    GDALDatasetH  hDataset;

    GDALAllRegister();

    hDataset = GDALOpen( pszFilename, GA_ReadOnly );
    if( hDataset == NULL )
    {
        ...;
    }
```

In Python:

```
    import gdal
    from gdalconst import *

    dataset = gdal.Open( filename, GA_ReadOnly )
    if dataset is None:
        ...
```

Note that if GDALOpen() returns NULL it means the open failed, and that an error messages will already have been emitted via CPLError(). If you want to control how errors are reported to the user review the CPLError() documentation. Generally speaking all of GDAL uses CPLError() for error reporting. Also, note that pszFilename need not actually be the name of a physical file (though it usually is). It's interpretation is driver dependent, and it might be an URL, a filename with additional parameters added at the end controlling the open or almost anything. Please try not to limit GDAL file selection dialogs to only selecting physical files.

13.2 Getting Dataset Information

As described in the GDAL Data Model, a GDALDataset contains a list of raster bands, all pertaining to the same area, and having the same resolution. It also has metadata, a coordinate system, a georeferencing transform, size of raster and various other information.

```
adfGeoTransform[0] /* top left x */
adfGeoTransform[1] /* w-e pixel resolution */
adfGeoTransform[2] /* rotation, 0 if image is "north up" */
adfGeoTransform[3] /* top left y */
adfGeoTransform[4] /* rotation, 0 if image is "north up" */
adfGeoTransform[5] /* n-s pixel resolution */
```

If we wanted to print some general information about the dataset we might do the following:

In C++:

```
double        adfGeoTransform[6];

printf( "Driver: %s/%s\n",
        poDataset->GetDriver()->GetDescription(),
        poDataset->GetDriver()->GetMetadataItem( GDAL_DMD_LONGNAME ) );

printf( "Size is %dx%dx%d\n",
        poDataset->GetRasterXSize(), poDataset->GetRasterYSize(),
        poDataset->GetRasterCount() );

if( poDataset->GetProjectionRef()  != NULL )
    printf( "Projection is `%s'\n", poDataset->GetProjectionRef() );

if( poDataset->GetGeoTransform( adfGeoTransform ) == CE_None )
{
    printf( "Origin = (%.6f,%.6f)\n",
            adfGeoTransform[0], adfGeoTransform[3] );

    printf( "Pixel Size = (%.6f,%.6f)\n",
            adfGeoTransform[1], adfGeoTransform[5] );
}
```

In C:

```
GDALDriverH   hDriver;
double        adfGeoTransform[6];

hDriver = GDALGetDatasetDriver( hDataset );
printf( "Driver: %s/%s\n",
        GDALGetDriverShortName( hDriver ),
        GDALGetDriverLongName( hDriver ) );

printf( "Size is %dx%dx%d\n",
        GDALGetRasterXSize( hDataset ),
        GDALGetRasterYSize( hDataset ),
        GDALGetRasterCount( hDataset ) );

if( GDALGetProjectionRef( hDataset ) != NULL )
    printf( "Projection is `%s'\n", GDALGetProjectionRef( hDataset ) );

if( GDALGetGeoTransform( hDataset, adfGeoTransform ) == CE_None )
{
    printf( "Origin = (%.6f,%.6f)\n",
            adfGeoTransform[0], adfGeoTransform[3] );

    printf( "Pixel Size = (%.6f,%.6f)\n",
```

```
            adfGeoTransform[1], adfGeoTransform[5] );
}
```

In Python:

```
print 'Driver: ', dataset.GetDriver().ShortName,'/', \
      dataset.GetDriver().LongName
print 'Size is ',dataset.RasterXSize,'x',dataset.RasterYSize, \
      'x',dataset.RasterCount
print 'Projection is ',dataset.GetProjection()

geotransform = dataset.GetGeoTransform()
if not geotransform is None:
    print 'Origin = (',geotransform[0], ',',geotransform[3],')'
    print 'Pixel Size = (',geotransform[1], ',',geotransform[5],')'
```

13.3 Fetching a Raster Band

At this time access to raster data via GDAL is done one band at a time. Also, there is metadata, blocksizes, color tables, and various other information available on a band by band basis. The following codes fetches a GDALRasterBand object from the dataset (numbered 1 through GetRasterCount()) and displays a little information about it.

In C++:

```
GDALRasterBand  *poBand;
int             nBlockXSize, nBlockYSize;
int             bGotMin, bGotMax;
double          adfMinMax[2];

poBand = poDataset->GetRasterBand( 1 );
poBand->GetBlockSize( &nBlockXSize, &nBlockYSize );
printf( "Block=%dx%d Type=%s, ColorInterp=%s\n",
        nBlockXSize, nBlockYSize,
        GDALGetDataTypeName(poBand->GetRasterDataType()),
        GDALGetColorInterpretationName(
            poBand->GetColorInterpretation()) );

adfMinMax[0] = poBand->GetMinimum( &bGotMin );
adfMinMax[1] = poBand->GetMaximum( &bGotMax );
if( ! (bGotMin && bGotMax) )
    GDALComputeRasterMinMax((GDALRasterBandH)poBand, TRUE, adfMinMax);

printf( "Min=%.3fd, Max=%.3f\n", adfMinMax[0], adfMinMax[1] );

if( poBand->GetOverviewCount() > 0 )
    printf( "Band has %d overviews.\n", poBand->GetOverviewCount() );

if( poBand->GetColorTable() != NULL )
    printf( "Band has a color table with %d entries.\n",
             poBand->GetColorTable()->GetColorEntryCount() );
```

In C:

```
GDALRasterBandH hBand;
int             nBlockXSize, nBlockYSize;
int             bGotMin, bGotMax;
double          adfMinMax[2];

hBand = GDALGetRasterBand( hDataset, 1 );
GDALGetBlockSize( hBand, &nBlockXSize, &nBlockYSize );
```

```
printf( "Block=%dx%d Type=%s, ColorInterp=%s\n",
        nBlockXSize, nBlockYSize,
        GDALGetDataTypeName(GDALGetRasterDataType(hBand)),
        GDALGetColorInterpretationName(
            GDALGetRasterColorInterpretation(hBand)) );

adfMinMax[0] = GDALGetRasterMinimum( hBand, &bGotMin );
adfMinMax[1] = GDALGetRasterMaximum( hBand, &bGotMax );
if( ! (bGotMin && bGotMax) )
    GDALComputeRasterMinMax( hBand, TRUE, adfMinMax );

printf( "Min=%.3fd, Max=%.3f\n", adfMinMax[0], adfMinMax[1] );

if( GDALGetOverviewCount(hBand) > 0 )
    printf( "Band has %d overviews.\n", GDALGetOverviewCount(hBand));

if( GDALGetRasterColorTable( hBand ) != NULL )
    printf( "Band has a color table with %d entries.\n",
             GDALGetColorEntryCount(
                 GDALGetRasterColorTable( hBand ) ) );
```

In Python (note several bindings are missing):

```
band = dataset.GetRasterBand(1)

print 'Band Type=',gdal.GetDataTypeName(band.DataType)

min = band.GetMinimum()
max = band.GetMaximum()
if min is None or max is None:
    (min,max) = band.ComputeRasterMinMax(1)
print 'Min=%.3f, Max=%.3f' % (min,max)

if band.GetOverviewCount() > 0:
    print 'Band has ', band.GetOverviewCount(), ' overviews.'

if not band.GetRasterColorTable() is None:
    print 'Band has a color table with ', \
    band.GetRasterColorTable().GetCount(), ' entries.'
```

13.4 Reading Raster Data

There are a few ways to read raster data, but the most common is via the GDALRasterBand::RasterIO() method. This method will automatically take care of data type conversion, up/down sampling and windowing. The following code will read the first scanline of data into a similarly sized buffer, converting it to floating point as part of the operation.

In C++:

```
float *pafScanline;
int   nXSize = poBand->GetXSize();

pafScanline = (float *) CPLMalloc(sizeof(float)*nXSize);
poBand->RasterIO( GF_Read, 0, 0, nXSize, 1,
                  pafScanline, nXSize, 1, GDT_Float32,
                  0, 0 );
```

In C:

```
float *pafScanline;
int   nXSize = GDALGetRasterBandXSize( hBand );
```

```
pafScanline = (float *) CPLMalloc(sizeof(float)*nXSize);
GDALRasterIO( hBand, GF_Read, 0, 0, nXSize, 1,
              pafScanline, nXSize, 1, GDT_Float32,
              0, 0 );
```

In Python:

```
scanline = band.ReadRaster( 0, 0, band.XSize, 1, \
                            band.XSize, 1, GDT_Float32 )
```

Note that the returned scanline is of type string, and contains xsize∗4 bytes of raw binary floating point data. This can be converted to Python values using the **struct** module from the standard library:

```
import struct

tuple_of_floats = struct.unpack('f' * b2.XSize, scanline)
```

The RasterIO call takes the following arguments.

```
CPLErr GDALRasterBand::RasterIO( GDALRWFlag eRWFlag,
                                 int nXOff, int nYOff, int nXSize, int nYSize,
                                 void * pData, int nBufXSize, int nBufYSize,
                                 GDALDataType eBufType,
                                 int nPixelSpace,
                                 int nLineSpace )
```

Note that the same RasterIO() call is used to read, or write based on the setting of eRWFlag (either GF_Read or GF_Write). The nXOff, nYOff, nXSize, nYSize argument describe the window of raster data on disk to read (or write). It doesn't have to fall on tile boundaries though access may be more efficient if it does.

The pData is the memory buffer the data is read into, or written from. It's real type must be whatever is passed as eBufType, such as GDT_Float32, or GDT_Byte. The RasterIO() call will take care of converting between the buffer's data type and the data type of the band. Note that when converting floating point data to integer RasterIO() rounds down, and when converting source values outside the legal range of the output the nearest legal value is used. This implies, for instance, that 16bit data read into a GDT_Byte buffer will map all values greater than 255 to 255, **the data is not scaled!**

The nBufXSize and nBufYSize values describe the size of the buffer. When loading data at full resolution this would be the same as the window size. However, to load a reduced resolution overview this could be set to smaller than the window on disk. In this case the RasterIO() will utilize overviews to do the IO more efficiently if the overviews are suitable.

The nPixelSpace, and nLineSpace are normally zero indicating that default values should be used. However, they can be used to control access to the memory data buffer, allowing reading into a buffer containing other pixel interleaved data for instance.

13.5 Closing the Dataset

Please keep in mind that GDALRasterBand objects are *owned* by their dataset, and they should never be destroyed with the C++ delete operator. GDALDataset's can be closed either by calling GDALClose() or using the delete operator on the GDALDataset. Either will result in proper cleanup, and flushing of any pending writes.

13.6 Techniques for Creating Files

New files in GDAL supported formats may be created if the format driver supports creation. There are two general techniques for creating files, using CreateCopy() and Create(). The CreateCopy method involves calling the CreateCopy() method on the format driver, and passing in a source dataset that should be copied. The Create method involves calling the Create() method on the driver, and then explicitly writing all the metadata, and raster data with separate calls. All drivers that support creating new files support the CreateCopy() method, but only a few support the Create() method.

To determine if a particular format supports Create or CreateCopy it is possible to check the DCAP_CREATE and DCAP_CREATECOPY metadata on the format driver object. Ensure that GDALAllRegister() has been called before calling GetDriverByName(). In this example we fetch a driver, and determine whether it supports Create() and/or CreateCopy().

In C++:

```
#include "cpl_string.h"
...
    const char *pszFormat = "GTiff";
    GDALDriver *poDriver;
    char **papszMetadata;

    poDriver = GetGDALDriverManager()->GetDriverByName(pszFormat);

    if( poDriver == NULL )
        exit( 1 );

    papszMetadata = poDriver->GetMetadata();
    if( CSLFetchBoolean( papszMetadata, GDAL_DCAP_CREATE, FALSE ) )
        printf( "Driver %s supports Create() method.\n", pszFormat );
    if( CSLFetchBoolean( papszMetadata, GDAL_DCAP_CREATECOPY, FALSE ) )
        printf( "Driver %s supports CreateCopy() method.\n", pszFormat );
```

In C:

```
#include "cpl_string.h"
...
    const char *pszFormat = "GTiff";
    GDALDriverH hDriver = GDALGetDriverByName( pszFormat );
    char **papszMetadata;

    if( hDriver == NULL )
        exit( 1 );

    papszMetadata = GDALGetMetadata( hDriver, NULL );
    if( CSLFetchBoolean( papszMetadata, GDAL_DCAP_CREATE, FALSE ) )
        printf( "Driver %s supports Create() method.\n", pszFormat );
    if( CSLFetchBoolean( papszMetadata, GDAL_DCAP_CREATECOPY, FALSE ) )
        printf( "Driver %s supports CreateCopy() method.\n", pszFormat );
```

In Python:

```
    format = "GTiff"
    driver = gdal.GetDriverByName( format )
    metadata = driver.GetMetadata()
    if metadata.has_key(gdal.DCAP_CREATE) \
       and metadata[gdal.DCAP_CREATE] == 'YES':
        print 'Driver %s supports Create() method.' % format
    if metadata.has_key(gdal.DCAP_CREATECOPY) \
       and metadata[gdal.DCAP_CREATECOPY] == 'YES':
        print 'Driver %s supports CreateCopy() method.' % format
```

Note that a number of drivers are read-only and won't support Create() or CreateCopy().

13.7 Using CreateCopy()

The GDALDriver::CreateCopy() method can be used fairly simply as most information is collected from the source dataset. However, it includes options for passing format specific creation options, and for reporting progress to the user as a long dataset copy takes place. A simple copy from the a file named pszSrcFilename, to a new file named pszDstFilename using default options on a format whose driver was previously fetched might look like this:

In C++:

```
GDALDataset *poSrcDS =
   (GDALDataset *) GDALOpen( pszSrcFilename, GA_ReadOnly );
GDALDataset *poDstDS;

poDstDS = poDriver->CreateCopy( pszDstFilename, poSrcDS, FALSE,
                                NULL, NULL, NULL );
if( poDstDS != NULL )
    delete poDstDS;
```

In C:

```
GDALDatasetH hSrcDS = GDALOpen( pszSrcFilename, GA_ReadOnly );
GDALDatasetH hDstDS;

hDstDS = GDALCreateCopy( hDriver, pszDstFilename, hSrcDS, FALSE,
                         NULL, NULL, NULL );
if( hDstDS != NULL )
    GDALClose( hDstDS );
```

In Python:

```
src_ds = gdal.Open( src_filename )
dst_ds = driver.CreateCopy( dst_filename, src_ds, 0 )
```

Note that the CreateCopy() method returns a writeable dataset, and that it must be closed properly to complete writing and flushing the dataset to disk. In the Python case this occurs automatically when "dst_ds" goes out of scope. The FALSE (or 0) value used for the bStrict option just after the destination filename in the CreateCopy() call indicates that the CreateCopy() call should proceed without a fatal error even if the destination dataset cannot be created to exactly match the input dataset. This might be because the output format does not support the pixel datatype of the input dataset, or because the destination cannot support writing georeferencing for instance.

A more complex case might involve passing creation options, and using a predefined progress monitor like this:

In C++:

```
#include "cpl_string.h"
...
    char **papszOptions = NULL;

    papszOptions = CSLSetNameValue( papszOptions, "TILED", "YES" );
    papszOptions = CSLSetNameValue( papszOptions, "COMPRESS", "PACKBITS" );
    poDstDS = poDriver->CreateCopy( pszDstFilename, poSrcDS, FALSE,
                                    papszOptions, GDALTermProgress, NULL );
    if( poDstDS != NULL )
        delete poDstDS;
    CSLDestroy( papszOptions );
```

In C:

```
#include "cpl_string.h"
...
    char **papszOptions = NULL;

    papszOptions = CSLSetNameValue( papszOptions, "TILED", "YES" );
    papszOptions = CSLSetNameValue( papszOptions, "COMPRESS", "PACKBITS" );
    hDstDS = GDALCreateCopy( hDriver, pszDstFilename, hSrcDS, FALSE,
                             papszOptions, GDALTermProgres, NULL );
    if( hDstDS != NULL )
        GDALClose( hDstDS );
    CSLDestroy( papszOptions );
```

In Python:

```
    src_ds = gdal.Open( src_filename )
    dst_ds = driver.CreateCopy( dst_filename, src_ds, 0,
                                [ 'TILED=YES', 'COMPRESS=PACKBITS' ] )
```

13.8 Using Create()

For situations in which you are not just exporting an existing file to a new file, it is generally necessary to use the GDALDriver::Create() method (though some interesting options are possible through use of virtual files or in-memory files). The Create() method takes an options list much like CreateCopy(), but the image size, number of bands and band type must be provided explicitly.

In C++:

```
    GDALDataset *poDstDS;
    char **papszOptions = NULL;

    poDstDS = poDriver->Create( pszDstFilename, 512, 512, 1, GDT_Byte,
                                papszOptions );
```

In C:

```
    GDALDatasetH hDstDS;
    char **papszOptions = NULL;

    hDstDS = GDALCreate( hDriver, pszDstFilename, 512, 512, 1, GDT_Byte,
                         papszOptions );
```

In Python:

```
    dst_ds = driver.Create( dst_filename, 512, 512, 1, gdal.GDT_Byte )
```

Once the dataset is successfully created, all appropriate metadata and raster data must be written to the file. What this is will vary according to usage, but a simple case with a projection, geotransform and raster data is covered here.

In C++:

```
    double adfGeoTransform[6] = { 444720, 30, 0, 3751320, 0, -30 };
    OGRSpatialReference oSRS;
    char *pszSRS_WKT = NULL;
    GDALRasterBand *poBand;
    GByte abyRaster[512*512];
```

```
poDstDS->SetGeoTransform( adfGeoTransform );

oSRS.SetUTM( 11, TRUE );
oSRS.SetWellKnownGeogCS( "NAD27" );
oSRS.exportToWkt( &pszSRS_WKT );
poDstDS->SetProjection( pszSRS_WKT );
CPLFree( pszSRS_WKT );

poBand = poDstDS->GetRasterBand(1);
poBand->RasterIO( GF_Write, 0, 0, 512, 512,
                  abyRaster, 512, 512, GDT_Byte, 0, 0 );

delete poDstDS;
```

In C:

```
double adfGeoTransform[6] = { 444720, 30, 0, 3751320, 0, -30 };
OGRSpatialReferenceH hSRS;
char *pszSRS_WKT = NULL;
GDALRasterBandH hBand;
GByte abyRaster[512*512];

GDALSetGeoTransform( hDstDS, adfGeoTransform );

hSRS = OSRNewSpatialReference( NULL );
OSRSetUTM( hSRS, 11, TRUE );
OSRSetWellKnownGeogCS( hSRS, "NAD27" );
OSRExportToWkt( hSRS, &pszSRS_WKT );
OSRDestroySpatialReference( hSRS );

GDALSetProjection( hDstDS, pszSRS_WKT );
CPLFree( pszSRS_WKT );

hBand = GDALGetRasterBand( hDstDS, 1 );
GDALRasterIO( hBand, GF_Write, 0, 0, 512, 512,
              abyRaster, 512, 512, GDT_Byte, 0, 0 );

GDALClose( hDstDS );
```

In Python:

```
import Numeric, osr

dst_ds.SetGeoTransform( [ 444720, 30, 0, 3751320, 0, -30 ] )

srs = osr.SpatialReference()
srs.SetUTM( 11, 1 )
srs.SetWellKnownGeogCS( 'NAD27' )
dst_ds.SetProjection( srs.ExportToWkt() )

raster = Numeric.zeros( (512, 512) )
dst_ds.GetRasterBand(1).WriteArray( raster )
```

Chapter 14

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.
- 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 - Compute a raster proximity map.
- gdal_polygonize - Generate polygons from raster.
- gdal_sieve - Raster Sieve filter.
- gdal-config - Get options required to build software using GDAL.

14.1 Creating New Files

Access an existing file to read it is generally quite simple. Just indicate the name of the file or dataset on the 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-paci` See `this blog entry` for more reference.

14.2 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).

–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.

Chapter 15

gdalinfo

lists information about a raster dataset

15.1 SYNOPSIS

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

15.2 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.

-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 the ones.

-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

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.

15.3 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
```

Chapter 16

gdal_translate

converts raster data between different formats

16.1 SYNOPSIS

```
gdal_translate [--help-general]
       [-ot {Byte/Int16/UInt16/UInt32/Int32/Float32/Float64/
             CInt16/CInt32/CFloat32/CFloat64}] [-strict]
       [-of format] [-b band] [-expand {rgb|rgba}]
       [-outsize xsize[%] ysize[%]]
       [-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]*
       [-mo "META-TAG=VALUE"]* [-quiet] [-sds]
       [-co "NAME=VALUE"]*
       src_dataset dst_dataset
```

16.2 DESCRIPTION

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

-ot: ***type*** 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.

-expand ***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.

-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.

-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.

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

-quiet: 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.

***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.

16.3 EXAMPLE

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

Chapter 17

gdaladdo

builds or rebuilds overview images

17.1 SYNOPSIS

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

17.2 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,average,gauss,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).

***filename*:** The file to build overviews for.

***levels*:** A list of integral overview levels to build.

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. *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.

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. The GeoTIFF driver stores overviews internally to the file operated on, unless the -ro flag is specified.

External overviews created in TIFF format may be compressed using the COMPRESS_OVERVIEW configuration option. All compression methods, supported by the GeoTIFF driver, 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}.

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

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

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)

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

Create an external JPEG-compressed GeoTIFF overview file from a 3-band RGB dataset:

```
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
```

Chapter 18

gdaltindex

builds a shapefile as a raster tileindex

18.1 SYNOPSIS

```
gdaltindex [-tileindex field_name] [-write_absolute_path] [-skip_different_projection] index_file [gdal_fi
```

18.2 DESCRIPTION

This program builds a shapefile with a record for each input raster file, an attribute containing the filename, and a polygon geometry outlining the raster. This output is suitable for use with UMN 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.

18.3 EXAMPLE

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

Chapter 19

gdalbuildvrt

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

19.1 SYNOPSIS

```
gdalbuildvrt [-tileindex field_name] [-resolution {highest|lowest|average}]
             [-input_file_list my_liste.txt] output.vrt [gdalfile]*
```

19.2 DESCRIPTION

This program builds a VRT (Virtual Dataset) that is a mosaic of the list of input gdal datasets. The list of input gdal datasets can be specified at the end of the command line, or put in a text file (one filename per line) for very long lists, or it can be a MapServer tileindex (see gdaltindex utility). In the later case, all entries in the tile index will be added to the VRT.

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.

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: 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.

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

19.3 EXAMPLE

```
gdalbuildvrt doq_index.vrt doq/*.tif
gdalbuildvrt -input_file_list my_liste.txt doq_index.vrt
```

Chapter 20

gdal_contour

builds vector contour lines from a raster elevation model

20.1 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>
```

20.2 DESCRIPTION

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

-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".

20.3 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
```

Chapter 21

gdal_rasterize

burns vector polygons into a raster

21.1 SYNOPSIS

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

21.2 DESCRIPTION

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

-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 a polygon.

-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 (not yet implemented).

-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.

***src_datasource*:** Any OGR supported readable datasource.

***dst_filename*:** The GDAL supported output file. Must support update mode access. Currently gdal_rasterize cannot create new output files though that may be added eventually.

21.3 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
```

Chapter 22

rgb2pct.py

Convert a 24bit RGB image to 8bit paletted

22.1 SYNOPSIS

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

22.2 DESCRIPTION

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

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

-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.

Chapter 23

pct2rgb.py

Convert an 8bit paletted image to 24bit RGB

23.1 SYNOPSIS

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

23.2 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.

***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.

Chapter 24

gdaltransform

transforms coordinates

24.1 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]]
```

24.2 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.

24.3 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
```

24.4 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
```

Chapter 25

nearblack

convert nearly black/white borders to black

25.1 SYNOPSIS

```
nearblack [-white] [-near dist] [-nb non_black_pixels]
          [-o outfile] infile
```

25.2 DESCRIPTION

This utility will scan an image and try to set all pixels that are nearly black (or nearly white) around the collar to exactly 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 currently always created with the HFA driver (Erdas Imagine - .img)

-white: Search for nearly white (255) pixels instead of nearly black pixels.

-near *dist*: Select how far from black (or white) the pixel values can be and still considered near black (white). 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.

***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 (white) until at least "non_black_pixels" pixels that are more than "dist" gray levels away from black (white) have been encountered at which point the scan stops. The nearly black (white) pixels are set to black (white).

Note that this algorithm is only applied to horizontal scanlines, so a photo with an indentation in the top or bottom will not have that indentation identified. 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.

Chapter 26

gdal_merge.py

mosaics a set of images

26.1 SYNOPSIS

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

26.2 DESCRIPTION

This utility will automatically mosaic a set of images. All the images must be in the same coordinate system and have a matching number of bands, but they may be overlapping, and at different resolutions.

-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.

-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.

-init *value*: Pre-initialize the output file with this value. However, it is not marked as the nodata value in the output file.

-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.

Chapter 27

gdal2tiles.py

generates directory with TMS tiles, KMLs and simple web viewers

27.1 SYNOPSIS

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

27.2 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, and will only work if GDAL was built with Python support.

Chapter 28

gdal-config

determines various information about a GDAL installation

28.1 SYNOPSIS

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

28.2 DESCRIPTION

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

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

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

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

–version: Reports the GDAL version.

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

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

Chapter 29

gdal_retile.py

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

```
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}]
               -levels numberoflevels
               -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 ";"

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

Chapter 30

gdal_grid

creates regular grid from the scattered data

30.1 SYNOPSIS

```
Usage: gdal_grid [--help-general] [--formats]
    [-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]
    [-l layername]* [-where expression] [-sql select_statement]
    [-txe xmin xmax] [-tye ymin ymax] [-outsize xsize ysize]
    [-a algorithm[:parameter1=value1]*]    [-quiet]
    <src_datasource> <dst_filename>
```

30.2 DESCRIPTION

This program creates regular grid (raster) from the scattered data read from the OGR datasource. Input data will be interpolated to fill grid nodes with values, you can choose from various interpolation methods.

-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 intersecting the bounding box described by (xmin, ymin) - (xmax, ymax).

-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.

-quiet: Suppress progress monitor and other non-error output.

***src_datasource*:** Any OGR supported readable datasource.

***dst_filename*:** The GDAL supported output file.

30.3 INTERPOLATION ALGORITHMS

There are number of interpolation algorithms to choose from.

30.3.1 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).

30.3.2 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.

30.3.3 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).

30.4 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.

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).

30.5 READING COMMA SEPARATED VALUES

Often you have a text file with a list of comma separated XYZ values to work with (so called CSV file). You can easily use that kind of data source in 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.

30.6 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 GTif
```

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 -outs
```

Chapter 31

GDAL Grid Tutorial

31.1 Introduction to Gridding

Gridding is a process of creating a regular grid (or call it a raster image) from the scattered data. Typically you have a set of arbitrary scattered over the region of survey measurements and you would like to convert them into the regular grid for further processing and combining with other grids.

Figure 31.1: Scattered data gridding

This problem can be solved using data interpolation or approximation algorithms. But you are not limited by interpolation here. Sometimes you don't need to interpolate your data but rather compute some statistics or data metrics over the region. Statistics is valuable itself or could be used for better choosing the interpolation algorithm and parameters.

That is what GDAL Grid API is about. It helps you to interpolate your data (see Interpolation of the Scattered Data) or compute data metrics (see Data Metrics Computation).

There are two ways of using this interface. Programmatically it is available through the GDALGridCreate C function; for end users there is a gdal_grid utility. The rest of this document discusses details on algorithms and their parameters implemented in GDAL Grid API.

31.2 Interpolation of the Scattered Data

31.2.1 Inverse Distance to a Power

The Inverse Distance to a Power gridding method is a weighted average interpolator. You should supply the input arrays with the scattered data values including coordinates of every data point and output grid geometry. The function will compute interpolated value for the given position in output grid.

For every grid node the resulting value Z will be calculated using formula:

$$Z = \frac{\sum_{i=1}^{n} \frac{Z_i}{r_i^p}}{\sum_{i=1}^{n} \frac{1}{r_i^p}}$$

where

- Z_i is a known value at point i,
- r is a distance from the grid node to point i,
- p is a weighting power,
- n is a number of points in search ellipse".

In this method the weighting factor w is

$$w = \frac{1}{r^p}$$

See GDALGridInverseDistanceToAPowerOptions for the list of GDALGridCreate parameters and invdist for the list of gdal_grid options.

31.2.2 Moving Average

The Moving Average is a simple data averaging algorithm. It uses a moving window of elliptic form to search values and averages all data points within the window. Search ellipse can be rotated by specified angle, the center of ellipse located at the grid node. Also the minimum number of data points to average can be set, if there are not enough points in window, the grid node considered empty and will be filled with specified NODATA value.

Mathematically it can be expressed with the formula:

$$Z = \frac{\sum_{i=1}^{n} Z_i}{n}$$

where

- Z is a resulting value at the grid node,
- Z_i is a known value at point i,
- n is a number of points in search search ellipse.

See GDALGridMovingAverageOptions for the list of GDALGridCreate parameters and average for the list of gdal_grid options.

31.2.3 Nearest Neighbor

The Nearest Neighbor method doesn't perform any interpolation or smoothing, it just takes the value of nearest point found in grid node search ellipse and returns it as a result. If there are no points found, the specified NODATA value will be returned.

See GDALGridNearestNeighborOptions for the list of GDALGridCreate parameters and nearest for the list of gdal_grid options.

31.3 Data Metrics Computation

All the metrics have the same set controlling options. See the GDALGridDataMetricsOptions.

31.3.1 Minimum Data Value

Minimum value found in grid node search ellipse. If there are no points found, the specified NODATA value will be returned.

$$Z = \min\left(Z_1, Z_2, \ldots, Z_n\right)$$

where

- Z is a resulting value at the grid node,
- Z_i is a known value at point i,
- n is a number of points in search ellipse".

31.3.2 Maximum Data Value

Maximum value found in grid node search ellipse. If there are no points found, the specified NODATA value will be returned.

$$Z = \max(Z_1, Z_2, \ldots, Z_n)$$

where

- Z is a resulting value at the grid node,
- Z_i is a known value at point i,
- n is a number of points in search ellipse".

31.3.3 Data Range

A difference between the minimum and maximum values found in grid node search ellipse. If there are no points found, the specified NODATA value will be returned.

$$Z = \max(Z_1, Z_2, \ldots, Z_n) - \min(Z_1, Z_2, \ldots, Z_n)$$

where

- Z is a resulting value at the grid node,
- Z_i is a known value at point i,
- n is a number of points in search ellipse".

31.4 Search Ellipse

Search window in gridding algorithms specified in the form of rotated ellipse. It is described by the three parameters:

- $radius_1$ is the first radius (x axis if rotation angle is 0),
- $radius_2$ is the second radius (y axis if rotation angle is 0),
- $angle$ is a search ellipse rotation angle (rotated counter clockwise).

Figure 31.2: Search ellipse

Only points located inside the search ellipse (including its border line) will be used for computation.

Chapter 32

Sponsoring GDAL/OGR

Development and maintenance of GDAL/OGR is supported by organizations contracting developers, organizations contributing improvements, users contributing improvements, and volunteers. Generally speaking this works well, and GDAL/OGR has improved substantially over the years.

However, there are still many tasks which do not receive the attention they should. Processing bug reports, writing documentation, writing test scripts, evaluating test script failures and user support often receive less attention than would be desired. Some new features of broad interest are not implemented because they aren't important enough to any one person or organization.

In order to provide sustained funding to support the maintenance, improvement and promotion of the GDAL/OGR project, the project seeks project sponsors to provide financial support. Sponsorship would be accomplished via the OSGeo Project Sponsorship program. Funds are held by OSGeo for disposition on behalf of the project, and dispersed at the discretion of the GDAL/OGR Project Steering Committee.

32.1 Sponsorship Uses

The primary intended use of the sponsorship funds is to hire a maintainer on a contract basis. The responsibilities would include:

- Addressing bug reports - reproducing then fixing or passing on to another developer.
- Extending, and running the test suite.
- Improving documentation.
- Other improvements to the software.
- General user support on the mailing list.

Sponsorship funds may also be used to contract for specific improvements to GDAL, provision of resources such as web hosting, funding code sprints, or funding project promotion. Decisions on spending of sponsorship funds will be made by the GDAL/OGR Project Steering Committee.

32.2 Sponsorship Benefits

Sponsoring GDAL/OGR provides the following benefits:

1. Ensures the sustainability and health of the GDAL/OGR project.
2. All sponsors will be listed on the project Credits page, ordered by contribution class (Platinum, Gold, Silver) with a link back to the sponsor. Silver sponsors and above may include a logo. Platinum sponsors may also have a logo appearing on the OSGeo main page.
3. Sponsors will be permitted to indicate they are project sponsors in web and other promotional materials, and use the GDAL/OGR logo.
4. Sponsor input on project focus and direction will be solicited via a survey.
5. Sponsors will received a degree of priority in processing of bug reports by any maintainer hired with sponsorship funds.
6. Sponsors will receive a detailed report annually on the use of sponsorship funds.

32.3 Sponsorship Process

Sponsors can sponsor GDAL for any amount of money of at least $500 USD. At or above the following levels a sponsor will be designated as being one of the following class:

1. $27000+ USD: Platinum Sponsor
2. $9000+ USD: Gold Sponsor
3. $3000+ USD: Silver Sponsor

Sponsorships last one year, after which they may be continuing with a new payment, or allowed to lapse. OSGeo is planning to be US 501(c)3 charity and sponsorships will be eligible as a charitable contribution for US taxpayers. Appropriate receipts can be issued when needed.

Organizations or individuals interested in sponsoring the GDAL/OGR project should contact Frank Warmerdam (warmerdam@pobox.com, +1 613 754 2041) with questions, or to make arrangements.

Chapter 33

GDAL VB6 Bindings Tutorial

33.1 Introduction

A partial set of Visual Basic 6 bindings have been build for GDAL. Internally these bindings use Declare based calls into the GDAL DLL C API but a set of shadow classes are also provided to provide object oriented access to GDAL services in VB6 similar to those provided in C++.

Note that the VB6 bindings are nowhere near comprehensive, nor are they documented. However, in combination with the corresponding C++ class documentation, and the following docs, it should be possible to use GDAL to accomplish a variety of operations. It is not believed that the VB6 bindings will be of any utility with earlier version of VB nor with VB.Net.

The classes for which access has been implemented includes GDALDriver, GDALDataset, GDALRasterBand, GDALColorTable, OGRSpatialReference and OGRCoordinateTransformation.

A mailing list specifically on VB6 GDAL topics has been setup at http://groups.yahoo.com/group/gdal-vb6-appdev .

33.2 Using GDAL VB6 Classes

To use VB6 GDAL bindings it is necessary to ensure that GDAL has been built with appropriate C entry points exported using the "stdcall" calling convention. This is the current default, but was not as recently as GDAL 1.2.6. So ensure you get a version more recent than 1.2.6.

Then add the GDAL VB6 class and module files to your VB6 project. These come from the gdal/vb6 directory and include the following key files:

- GDAL.bas - The main user visible module.
- GDALCore.bas - This module is for internal use.
- GDALDriver.cls - The GDALDriver class.
- GDALDataset.cls - The GDALDataset class.
- GDALRasterBand.cls - The GDALRasterBand class.
- GDALColorTable.cls - The GDALColorTable class.
- OGRSpatialReference.cls - The OGRSpatialReference class.
- OGRCoordinateTransformation.cls - The OGRCoordinateTransformation class.

You may need to edit GDALCore.bas, and change occurrences of gdal12.dll to match what your GDAL DLL is called. You can include a full path to the DLL if it can't be guaranteed to be in the current working directory of the application (or the windows system32 directory).

You should also be able to load the "test" project from the gdal\vb6\test directory. The test project has test menu items roughly corresponding to the tasks in the following tutorial topics.

33.3 Tutorial - Read Dataset

This brief tutorial will demonstrate open a GDAL file, and fetching out some information, about the dataset, and the individual bands. The results are printed to the default from in the following example for simplicity.

Before opening the file we need to register the GDAL format drivers. Normally we will just register all the drivers with GDALAllRegister().

```
Call GDAL.AllRegister()
```

Then we need to try and open the dataset. The GDAL.OpenDS() function returns a GDALDataset object, so we dimension an appropriate object for this. GDAL.OpenDS() is the VB6 equivalent of the GDALDataset::GDALOpen() function.

```
Dim ds As GDALDataset

Set ds = GDAL.OpenDS( "utm.tif", GDAL.GA_ReadOnly )
```

Then we need to check if the open succeeded, and if not report an error.

```
If not ds.IsValid() Then
  Call MsgBox( "Open failed: " & GDAL.GetLastErrorMsg() )
  Exit Sub
End If
```

If things succeeded, we query width of the image in pixels (XSize), Height of the image in pixels (YSize) and number of bands (BandCount) from the dataset properties.

```
Print "Size: " & ds.XSize & "x" & ds.YSize & "x" & ds.BandCount
```

Next we read metadata from the dataset using the VB6 equivalent of the GDALMajorObject::GetMetadata() method, and report it to the user. Metadata is returned as an array of strings of "name=value" items. Array indices start at zero in the returned array. The domain argument should normally be vbNullString though in specialized circumstances other domains might apply.

```
Dim MD As Variant
MD = ds.GetMetadata(vbNullString)
If (UBound(MD) > 0) Then
  Print "Metadata:"
  For i = 1 To UBound(MD)
    Print "  " & MD(i)
  Next i
End If
```

Parsing the "name=value" strings from GetMetadata() can be a bit of a bother, so if we were looking for specific values we could use GetMetadataItem() and provide a specific item we want to extract. This would extract just the value if it is found, or an empty string otherwise. The GetMetadataItem() is an analog of the C++ GDALMajorObject::GetMetadataItem() method.

```
Dim MDValue As String

MDValue = ds.GetMetadataItem( "TIFF_DATETIME", vbNullString )
if MDValue <> "" Then
  Print "Creation Date: " & MDValue
End If
```

The GDALDataset::GetGeoTransform() method is used to get fetch the affine transformation used to relate pixel/line locations on the image to georeferenced locations in the current coordinate system. In the most common case (image is not rotated or sheared) you can just report the origin (upper left corner) and pixel size from these values. The method returns 0 on success or an error class if it fails, so we only use the return result (placed into the Geotransform array) on success.

```
Dim Geotransform(6) As Double
```

```
If ds.GetGeoTransform( Geotransform ) = 0 Then
  If Geotransform(2) = 0 and Geotransform(4) = 0 Then
    Print "Origin: " & Geotransform(0) & "," & Geotransform(3)
    Print "Pixel Size: " & Geotransform(1) & "x" & (-1 * Geotransform(5))
  End If
End If
```

The coordinate system can be fetched using the GDALDataset::GetProjectionRef() analog, GDALDataset.GetProjection(). The returned string is in OpenGIS Well Known Text format. A later example will show how to use an OGRSpatialReference object to reformat the WKT into more readable format and make other use of it.

```
Dim WKT As String

WKT = ds.GetProjection()
If Len(WKT) > 0 Then
  Print "Projection: " & WKT
End If
```

GDALDataset objects have one or more raster bands associated with them. GDALRasterBand objects can have metadata (accessed the same as on the GDALDataset) as well as an array of pixel values, and various specialized metadata items like data type, color interpretation, offset/scale. Here we report a few of the items.

First we loop over all the bands, fetching a band object for each band and report the band number, and block size.

```
For i = 1 To ds.BandCount
  Dim band As GDALRasterBand

  Set band = ds.GetRasterBand(i)
  Print "Band " & i & " BlockSize: " & band.BlockXSize & "x" & band.BlockYSize
```

The GDALRasterBand has a DataType property which has the value returned by the C++ method GDALRasterBand::GetRasterDataType(). The returned value is an integer, but may be compared to the predefined constants GDAL.GDT_Byte, GDAL.GDT_UInt16, GDAL.GDT_Int16, GDAL.GDT_UInt32, GDAL.GDT_Int32, GDAL.GDT_Float32, GDAL.GDT_Float64, GDAL.GDT_CInt16, GDAL.GDT_CInt32, GDAL.GDT_CFloat32 and GDAL.GDT_CFloat64. In this case we use the GDAL.GetDataTypeName() method to convert the data type into a name we can show the user.

```
  Print "     DataType=" & GDAL.GetDataTypeName(band.DataType) _
```

We also report the offset, scale, minimum and maximum for the band.

```
  Print " Offset=" & band.GetOffset() & " Scale=" & band.GetScale() _
      & " Min=" & band.GetMinimum() & " Max=" & band.GetMaximum()
```

GDALRasterBands can also have GDALColorTable objects associated with them. They are read with the GDALRasterBand::GetColorTable() analog in VB6. Individual RGBA entries should be read into a 4 Integer array.

```
  Dim ct As GDALColorTable
  Set ct = band.GetColorTable()
  If ct.IsValid() Then
    Dim CEntry(4) As Integer
    Print "    Has Color Table, " & ct.EntryCount & " entries"
    For iColor = 0 To ct.EntryCount - 1
```

```
    Call ct.GetColorEntryAsRGB(iColor, CEntry)
    Print "      " & iColor & ": " & CEntry(0) & "," & CEntry(1) & "," & CEntry(2) & "," & CEntry(3)
  Next iColor
End If
```

But of course, the most important contents of a GDAL file is the raster pixel values themselves. The C++ GDALRasterBand::RasterIO() method is provided in a somewhat simplified form. A predimensioned 1D or 2D array of type Byte, Int, Long, Float or Double is passed to the RasterIO() method along with the band and window to be read. Internally the "buffer size" and datatype is extracted from the dimensions of the passed in buffer.

This example dimensions the RawData array to be the size of one scanline of data (XSize x 1) and reads the first whole scanline of data from the file, but only prints out the second and tenth values (since the buffer indexes are zero based).

```
Dim err As Long
Dim RawData() As Double
ReDim RawData(ds.XSize) As Double

err = band.RasterIO(GDAL.GF_Read, 0, 0, ds.XSize, 1, RawData)
if err = 0 Then
  Print "    Data: " & RawData(1) & " " & RawData(9)
End If
```

Finally, when done accessing a GDALDataset we can explicitly close it using the CloseDS() method, or just let it fall out of scope in which case it will be closed automatically.

```
Call ds.CloseDS()
```

33.4 Tutorial - Creating Files

Next we address creating a new file from an existing file. To create a new file, you have to select a GDALDriver to do the creating. The GDALDriver is essentially an object representing a file format. We fetch it with the GetDriverByName() call from the GDAL module using the driver name.

```
Dim Drv As GDALDriver

Call GDAL.AllRegister
Drv = GDALCore.GetDriverByName( "GTiff" )
If Not Drv.IsValid() Then
    Call MsgBox( "GTiff driver not found " )
    Exit Sub
End If
```

You could get a list of registered drivers, and identify which support creation something like this:

```
drvCount = GDAL.GetDriverCount
For drvIndex = 0 To drvCount - 1
    Set Drv = GDAL.GetDriver(drvIndex)
    If Drv.GetMetadataItem(GDAL.DCAP_CREATE, "") = "YES" _
        Or Drv.GetMetadataItem(GDAL.DCAP_CREATECOPY, "") = "YES" Then
        xMsg = " (Read/Write)"
    Else
        xMsg = " (ReadOnly)"
    End If

    Print Drv.GetShortName() & ": " & Drv.GetMetadataItem(GDAL.DMD_LONGNAME, "") & xMsg
Next drvIndex
```

Once we have the driver object, the simplest way of creating a new file is to use CreateCopy(). This tries to create a copy of the input file in the new format. A complete segment (without any error checking) would look like the following. The CreateCopy() method corresponds to the C++ method GDALDriver::CreateCopy(). The VB6 implementation does not support the use of progress callbacks.

```
Dim Drv As GDALDriver
Dim SrcDS As GDALDataset, DstDS As GDALDataset

Call GDAL.AllRegister
Set Drv = GDALCore.GetDriverByName( "GTiff" )

Set SrcDS = GDAL.Open( "in.tif", GDAL.GA_ReadOnly )
Set DstDS = Drv.CreateCopy( "out.tif", SrcDS, True, Nothing )
```

This is nice and simple, but sometimes we need to create a file with more detailed control. So, next we show how to create a file and then copy pieces of data to it "manually". The GDALDriver::Create() analog is Create().

```
Set DstDS = Drv.Create("out.tif", SrcDS.XSize, SrcDS.YSize, _
                       SrcDS.BandCount, GDAL.GDT_Byte, Nothing)
```

In some cases we may want to provide some creation options, which is demonstrated here. Creation options (like metadata set through the SetMetadata() method) are arrays of Strings.

```
Dim CreateOptions(1) As String

CreateOptions(1) = "PHOTOMETRIC=MINISWHITE"
Set DstDS = Drv.Create("out.tif", SrcDS.XSize, SrcDS.YSize, _
                       SrcDS.BandCount, GDAL.GDT_Byte, CreateOptions)
```

When copying the GeoTransform, we take care to check that reading the geotransform actually worked. Most methods which return CPLErr in C++ also return it in VB6. A return value of 0 will indicate success, and non-zero is failure.

```
Dim err As Long
Dim gt(6) As Double

err = SrcDS.GetGeoTransform(gt)
If err = 0 Then
    Call DstDS.SetGeoTransform(gt)
End If
```

Copy the projection. Even if GetProjection() fails we get an empty string which is safe enough to set on the target. Similarly for metadata.

```
Call DstDS.SetProjection(SrcDS.GetProjection())
Call DstDS.SetMetadata(SrcDS.GetMetadata(""), "")
```

Next we loop, processing bands, and copy some common data items.

```
For iBand = 1 To SrcDS.BandCount
    Dim SrcBand As GDALRasterBand, DstBand As GDALRasterBand

    Set SrcBand = SrcDS.GetRasterBand(iBand)
    Set DstBand = DstDS.GetRasterBand(iBand)

    Call DstBand.SetMetadata(SrcBand.GetMetadata(""), "")
    Call DstBand.SetOffset(SrcBand.GetOffset())
```

```
Call DstBand.SetScale(SrcBand.GetScale())

Dim NoDataValue As Double, Success As Long

NoDataValue = SrcBand.GetNoDataValue(Success)
If Success <> 0 Then
    Call DstBand.SetNoDataValue(NoDataValue)
End If
```

Then, if one is available, we copy the palette.

```
Dim ct As GDALColorTable
Set ct = SrcBand.GetColorTable()
If ct.IsValid() Then
    err = DstBand.SetColorTable(ct)
End If
```

Finally, the meat and potatoes. We copy the image data. We do this one scanline at a time so that we can support very large images without require large amounts of RAM. Here we use a Double buffer for the scanline, but if we knew in advance the type of the image, we could dimension a buffer of the appropriate type. The RasterIO() method internally knows how to convert pixel data types, so using Double ensures all data types (except for complex) are properly preserved, though at the cost of some extra data conversion internally.

```
Dim Scanline() As Double, iLine As Long
ReDim Scanline(SrcDS.XSize) As Double

' Copy band raster data.
For iLine = 0 To SrcDS.YSize - 1
    Call SrcBand.RasterIO(GDAL.GF_Read, 0, iLine, SrcDS.XSize, 1, _
                          Scanline)
    Call DstBand.RasterIO(GDAL.GF_Write, 0, iLine, SrcDS.XSize, 1, _
                          Scanline)
Next iLine
```

33.5 Tutorial - Coordinate Systems and Reprojection

The GDAL VB6 bindings also include limited support for use of the OGRSpatialReference and OGRCoordinateTransformation classes. The OGRSpatialReference represents a coordinate system and can be used to parse, manipulate and form WKT strings, such as those returned by the GDALDataset.GetProjection() method. The OGRCoordinateTransformation class provides a way of reprojecting between two coordinate systems.

The following example shows how to report the corners of an image in georeferenced and geographic (lat/long) coordinates. First, we open the file, and read the geotransform.

```
Dim ds As GDALDataset

Call GDALCore.GDALAllRegister
Set ds = GDAL.OpenDS(FileDlg.Filename, GDAL.GA_ReadOnly)

If ds.IsValid() Then
    Dim Geotransform(6) As Double

    Call ds.GetGeoTransform(Geotransform)
```

Next, we fetch the coordinate system, and if it is non-empty we try to instantiate an OGRSpatialReference from it.

```
        ' report projection in pretty format.
        Dim WKT As String
        Dim srs As New OGRSpatialReference
        Dim latlong_srs As OGRSpatialReference
        Dim ct As New OGRCoordinateTransformation

        WKT = ds.GetProjection()
        If Len(WKT) > 0 Then
            Print "Projection: "
            Call srs.SetFromUserInput(WKT)
```

If the coordinate system is projected it will have a PROJECTION node. In that case we build a new coordinate system which is the corresponding geographic coordinate system. So for instance if the "srs" was UTM 11 WGS84 then it's corresponding geographic coordinate system would just be WGS84. Once we have these two coordinate systems, we build a transformer to convert between them.

```
            If srs.GetAttrValue("PROJECTION", 0) <> "" Then
                Set latlong_srs = srs.CloneGeogCS()
                Set ct = GDAL.CreateCoordinateTransformation(srs, latlong_srs)
            End If
        End If
```

Next we call a helper function to report each corner, and the center. We pass in the name of the corner, the pixel/line location at the corner, and the geotransform and transformer object.

```
        Call ReportCorner("Top Left      ", 0, 0, _
                          Geotransform, ct)
        Call ReportCorner("Top Right     ", ds.XSize, 0, _
                          Geotransform, ct)
        Call ReportCorner("Bottom Left   ", 0, ds.YSize, _
                          Geotransform, ct)
        Call ReportCorner("Bottom Right  ", ds.XSize, ds.YSize, _
                          Geotransform, ct)
        Call ReportCorner("Center        ", ds.XSize / 2#, ds.YSize / 2#, _
                          Geotransform, ct)
```

The ReportCorner subroutine starts by computing the corresponding georeferenced x and y location using the pixel/line coordinates and the geotransform.

```
Private Sub ReportCorner(CornerName As String, pixel As Double, line As Double, _
                         gt() As Double, ct As OGRCoordinateTransformation)

    Dim geox As Double, geoy As Double

    geox = gt(0) + pixel * gt(1) + line * gt(2)
    geoy = gt(3) + pixel * gt(4) + line * gt(5)
```

Next, if we have a transformer, we use it to compute a corresponding latitude and longitude.

```
    Dim longitude As Double, latitude As Double, Z As Double
    Dim latlong_valid As Boolean

    latlong_valid = False

    If ct.IsValid() Then
        Z = 0
        longitude = geox
        latitude = geoy
        latlong_valid = ct.TransformOne(longitude, latitude, Z)
    End If
```

Then we report the corner location in georeferenced, and if we have it geographic coordinates.

```
    If latlong_valid Then
        Print CornerName & geox & "," & geoy & "    " & longitude & "," & latitude
    Else
        Print CornerName & geox & "," & geoy
    End If
End Sub
```

Chapter 34

GDAL Warp API Tutorial

34.1 Overview

The GDAL Warp API (declared in gdalwarper.h) provides services for high performance image warping using application provided geometric transformation functions (GDALTransformerFunc), a variety of resampling kernels, and various masking options. Files much larger than can be held in memory can be warped.

This tutorial demonstrates how to implement an application using the Warp API. It assumes implementation in C++ as C and Python bindings are incomplete for the Warp API. It also assumes familiarity with the GDAL Data Model, and the general GDAL API.

Applications normally perform a warp by initializing a GDALWarpOptions structure with the options to be utilized, instantiating a GDALWarpOperation based on these options, and then invoking the GDALWarpOperation::ChunkAndWarpImage() method to perform the warp options internally using the GDALWarpKernel class.

34.2 A Simple Reprojection Case

First we will construct a relatively simple example for reprojecting an image, assuming an appropriate output file already exists, and with minimal error checking.

```
#include "gdalwarper.h"

int main()
{
    GDALDatasetH  hSrcDS, hDstDS;

    // Open input and output files.

    GDALAllRegister();

    hSrcDS = GDALOpen( "in.tif", GA_ReadOnly );
    hDstDS = GDALOpen( "out.tif", GA_Update );

    // Setup warp options.

    GDALWarpOptions *psWarpOptions = GDALCreateWarpOptions();

    psWarpOptions->hSrcDS = hSrcDS;
    psWarpOptions->hDstDS = hDstDS;

    psWarpOptions->nBandCount = 1;
    psWarpOptions->panSrcBands =
        (int *) CPLMalloc(sizeof(int) * psWarpOptions->nBandCount );
    psWarpOptions->panSrcBands[0] = 1;
    psWarpOptions->panDstBands =
        (int *) CPLMalloc(sizeof(int) * psWarpOptions->nBandCount );
    psWarpOptions->panDstBands[0] = 1;

    psWarpOptions->pfnProgress = GDALTermProgress;

    // Establish reprojection transformer.

    psWarpOptions->pTransformerArg =
        GDALCreateGenImgProjTransformer( hSrcDS,
                                         GDALGetProjectionRef(hSrcDS),
                                         hDstDS,
                                         GDALGetProjectionRef(hDstDS),
                                         FALSE, 0.0, 1 );
    psWarpOptions->pfnTransformer = GDALGenImgProjTransform;
```

```
    // Initialize and execute the warp operation.

    GDALWarpOperation oOperation;

    oOperation.Initialize( psWarpOptions );
    oOperation.ChunkAndWarpImage( 0, 0,
                                  GDALGetRasterXSize( hDstDS ),
                                  GDALGetRasterYSize( hDstDS ) );

    GDALDestroyGenImgProjTransformer( psWarpOptions->pTransformerArg );
    GDALDestroyWarpOptions( psWarpOptions );

    GDALClose( hDstDS );
    GDALClose( hSrcDS );

    return 0;
}
```

This example opens the existing input and output files (in.tif and out.tif). A GDALWarpOptions structure is allocated (GDALCreateWarpOptions() sets lots of sensible defaults for stuff, always use it for defaulting things), and the input and output file handles, and band lists are set. The panSrcBands and panDstBands lists are dynamically allocated here and will be free automatically by GDALDestroyWarpOptions(). The simple terminal output progress monitor (GDALTermProgress) is installed for reporting completion progress to the user.

GDALCreateGenImgProjTransformer() is used to initialize the reprojection transformation between the source and destination images. We assume that they already have reasonable bounds and coordinate systems set. Use of GCPs is disabled.

Once the options structure is ready, a GDALWarpOperation is instantiated using them, and the warp actually performed with GDALWarpOperation::ChunkAndWarpImage(). Then the transformer, warp options and datasets are cleaned up.

Normally error check would be needed after opening files, setting up the reprojection transformer (returns NULL on failure), and initializing the warp.

34.3 Other Warping Options

The GDALWarpOptions structures contains a number of items that can be set to control warping behavior. A few of particular interest are:

1. GDALWarpOptions::dfWarpMemoryLimit - Set the maximum amount of memory to be used by the GDALWarpOperation when selecting a size of image chunk to operate on. The value is in bytes, and the default is likely to be conservative (small). Increasing the chunk size can help substantially in some situations but care should be taken to ensure that this size, plus the GDAL cache size plus the working set of GDAL, your application and the operating system are less than the size of RAM or else excessive swapping is likely to interfere with performance. On a system with 256MB of RAM, a value of at least 64MB (roughly 64000000 bytes) is reasonable. Note that this value does **not** include the memory used by GDAL for low level block caching.
2. GDALWarpOpations::eResampleAlg - One of GRA_NearestNeighbour (the default, and fastest), GRA_Bilinear (2x2 bilinear resampling) or GRA_Cubic. The GRA_NearestNeighbour type should generally be used for thematic or colormapped images. The other resampling types may give better results for thematic images, especially when substantially changing resolution.
3. GDALWarpOptions::padfSrcNoDataReal - This array (one entry per band being processed) may be setup with a "nodata" value for each band if you wish to avoid having pixels of some background value copied to the destination image.

4. GDALWarpOptions::papszWarpOptions - This is a string list of NAME=VALUE options passed to the warper. See the GDALWarpOptions::papszWarpOptions docs for all options. Supported values include:

    - INIT_DEST=[value] or INIT_DEST=NO_DATA: This option forces the destination image to be initialized to the indicated value (for all bands) or indicates that it should be initialized to the NO_DATA value in padfDstNoDataReal/padfDstNoDataImag. If this value isn't set the destination image will be read and the source warp overlayed on it.
    - WRITE_FLUSH=YES/NO: This option forces a flush to disk of data after each chunk is processed. In some cases this helps ensure a serial writing of the output data otherwise a block of data may be written to disk each time a block of data is read for the input buffer resulting in a lot of extra seeking around the disk, and reduced IO throughput. The default at this time is NO.

34.4 Creating the Output File

In the previous case an appropriate output file was already assumed to exist. Now we will go through a case where a new file with appropriate bounds in a new coordinate system is created. This operation doesn't relate specifically to the warp API. It is just using the transformation API.

```
#include "gdalwarper.h"
#include "ogr_spatialref.h"

...

    GDALDriverH hDriver;
    GDALDataType eDT;
    GDALDatasetH hDstDS;
    GDALDatasetH hSrcDS;

    // Open the source file.

    hSrcDS = GDALOpen( "in.tif", GA_ReadOnly );
    CPLAssert( hSrcDS != NULL );

    // Create output with same datatype as first input band.

    eDT = GDALGetRasterDataType(GDALGetRasterBand(hSrcDS,1));

    // Get output driver (GeoTIFF format)

    hDriver = GDALGetDriverByName( "GTiff" );
    CPLAssert( hDriver != NULL );

    // Get Source coordinate system.

    const char *pszSrcWKT, *pszDstWKT = NULL;

    pszSrcWKT = GDALGetProjectionRef( hSrcDS );
    CPLAssert( pszSrcWKT != NULL && strlen(pszSrcWKT) > 0 );

    // Setup output coordinate system that is UTM 11 WGS84.

    OGRSpatialReference oSRS;

    oSRS.SetUTM( 11, TRUE );
    oSRS.SetWellKnownGeogCS( "WGS84" );

    oSRS.exportToWkt( &pszDstWKT );

    // Create a transformer that maps from source pixel/line coordinates
```

```
// to destination georeferenced coordinates (not destination
// pixel line).  We do that by omitting the destination dataset
// handle (setting it to NULL).

void *hTransformArg;

hTransformArg =
    GDALCreateGenImgProjTransformer( hSrcDS, pszSrcWKT, NULL, pszDstWKT,
                                     FALSE, 0, 1 );
CPLAssert( hTransformArg != NULL );

// Get approximate output georeferenced bounds and resolution for file.

double adfDstGeoTransform[6];
int nPixels=0, nLines=0;
CPLErr eErr;

eErr = GDALSuggestedWarpOutput( hSrcDS,
                                GDALGenImgProjTransform, hTransformArg,
                                adfDstGeoTransform, &nPixels, &nLines );
CPLAssert( eErr == CE_None );

GDALDestroyGenImgProjTransformer( hTransformArg );

// Create the output file.

hDstDS = GDALCreate( hDriver, "out.tif", nPixels, nLines,
                     GDALGetRasterCount(hSrcDS), eDT, NULL );

CPLAssert( hDstDS != NULL );

// Write out the projection definition.

GDALSetProjection( hDstDS, pszDstWKT );
GDALSetGeoTransform( hDstDS, adfDstGeoTransform );

// Copy the color table, if required.

GDALColorTableH hCT;

hCT = GDALGetRasterColorTable( GDALGetRasterBand(hSrcDS,1) );
if( hCT != NULL )
    GDALSetRasterColorTable( GDALGetRasterBand(hDstDS,1), hCT );

... proceed with warp as before ...
```

Some notes on this logic:

- We need to create the transformer to output coordinates such that the output of the transformer is georeferenced, not pixel line coordinates since we use the transformer to map pixels around the source image into destination georeferenced coordinates.

- The GDALSuggestedWarpOutput() function will return an adfDstGeoTransform, nPixels and nLines that describes an output image size and georeferenced extents that should hold all pixels from the source image. The resolution is intended to be comparable to the source, but the output pixels are always square regardless of the shape of input pixels.

- The warper requires an output file in a format that can be "randomly" written to. This generally limits things to uncompressed formats that have an implementation of the Create() method (as opposed to CreateCopy()). To warp to compressed formats, or CreateCopy() style formats it is necessary to produce a full temporary copy of the image in a better behaved format, and then CreateCopy() it to the desired final format.

- The Warp API copies only pixels. All colormaps, georeferencing and other metadata must be copied to the destination by the application.

34.5 Performance Optimization

There are a number of things that can be done to optimize the performance of the warp API.

1. Increase the amount of memory available for the Warp API chunking so that larger chunks can be operated on at a time. This is the GDALWarpOptions::dfWarpMemoryLimit parameter. In theory the larger the chunk size operated on the more efficient the I/O strategy, and the more efficient the approximated transformation will be. However, the sum of the warp memory and the GDAL cache should be less than RAM size, likely around 2/3 of RAM size.
2. Increase the amount of memory for GDAL caching. This is especially important when working with very large input and output images that are scanline oriented. If all the input or output scanlines have to be re-read for each chunk they intersect performance may degrade greatly. Use GDALSetCacheMax() to control the amount of memory available for caching within GDAL.
3. Use an approximated transformation instead of exact reprojection for each pixel to be transformed. This code illustrates how an approximated transformation could be created based on a reprojection transformation, but with a given error threshold (dfErrorThreshold in output pixels).

```
hTransformArg =
    GDALCreateApproxTransformer( GDALGenImgProjTransform,
                                 hGenImgProjArg, dfErrorThreshold );
pfnTransformer = GDALApproxTransform;
```

4. When writing to a blank output file, use the INIT_DEST option in the GDALWarpOptions::papszWarpOptions to cause the output chunks to be initialized to a fixed value, instead of being read from the output. This can substantially reduce unnecessary IO work.
5. Use tiled input and output formats. Tiled formats allow a given chunk of source and destination imagery to be accessed without having to touch a great deal of extra image data. Large scanline oriented files can result in a great deal of wasted extra IO.
6. Process all bands in one call. This ensures the transformation calculations don't have to be performed for each band.
7. Use the GDALWarpOperation::ChunkAndWarpMulti() method instead of GDALWarpOperation::ChunkAndWarpImage(). It uses a separate thread for the IO and the actual image warp operation allowing more effective use of CPU and IO bandwidth. For this to work GDAL needs to have been built with multi-threading support (default on Win32, –with-pthreads on Unix).
8. The resampling kernels vary is work required from nearest neighbour being least, then bilinear then cubic. Don't use a more complex resampling kernel than needed.
9. Avoid use of esoteric masking options so that special simplified logic case be used for common special cases. For instance, nearest neighbour resampling with no masking on 8bit data is highly optimized compared to the general case.

34.6 Other Masking Options

The GDALWarpOptions include a bunch of esoteric masking capabilities, for validity masks, and density masks on input and output. Some of these are not yet implemented and others are implemented but poorly tested. Other than per-band validity masks it is advised that these features be used with caution at this time.

Chapter 35

GDAL for Windows CE

Overview
Features
Supported Platforms
Content of 'wince' directory
Building GDAL for Windows CE using Microsoft Visual C++ 2005
Enable PROJ.4 support
wince_building_geos
How can I help?

35.1 Overview

This document is devoted to give some overview of the GDAL port for Windows CE operating system.

35.2 Features

Currently, from version 1.4.0, GDAL includes following features for Windows CE platform:

- CPL library
- GDAL and OGR core API
- GDAL drivers:
  - AAIGrid
  - DTED
  - GeoTIFF
- OGR drivers:
  - Generic
  - CSV
  - MITAB
  - ESRI Shapefile
- Unit Test suite (gdalautotest/cpp)
- Optional PROJ.4 support
- Optional GEOS support

35.3 Supported Platforms

GDAL for Windows CE has been tested on following versions of Windows CE:

- Windows CE 3.x
  - Pocket PC 2002

- Windows CE 4.x
  - Windows Mobile 2003
- Windows CE 5.x
  - Windows Mobile 5
  - customized versions of Windows CE 5.0

Supported compilers for Windows CE operating system:

- Microsoft Visual C++ 2005 Standard, Professional or Team Suite Edition
- Microsoft eMbedded Visual C++ 4.0

  Note:

  Currently, no project files provided for eVC++ 4.0 IDE

35.4 Content of 'wince' directory

Note:

Due to problems with removing directories from CVS and missed synchronization of RC branch, the 'wince' directory includes a few deprecated project files (see below).
Please **DON'T USE** them, unless you want to fix them yourself.

Active content:

- **msvc80** - project for Visual C++ 2005 to build GDAL DLL for Windows CE
- README - the file you're currently reading
- TODO - planned and requested features

Deprecated

Following directories and projects are deprecated. **DON'T USE THEM!**

- evc4_gdalce_dll
- evc4_gdalce_dll_test
- evc4_gdalce_lib
- evc4_gdalce_lib_test
- msvc8_gdalce_lib
- msvc8_gdalce_lib_test
- wce_test_dll
- wce_test_lib
- wcelibcex

35.5 Building GDAL for Windows CE using Microsoft Visual C++ 2005

1. Requirements

   - You need to have installed Visual C++ 2005 Standard, Professional or Team Suite Edition.
   - You also need to have installed at least one SDK for Windows CE platform:
     - Windows Mobile 2003 Pocket PC SDK
     - Windows Mobile 2003 SmartphoneSDK
     - Windows Mobile 5.0 Pocket PC SDK
     - Windows Mobile 5.0 Smartphone SDK
   - Last requirement is the `Run-time Type Information library for the Pocket PC 2003 SDK`.

2. External dependencies

   There is only one external dependency required to build GDAL for Windows CE. This dependency is `WCELIBCEX` library available to download from:

   `http://sourceforge.net/projects/wcelibcex`

   You can download latest release - `wcelibcex-1.0` - or checkout sources directly form SVN. In both cases, you will be provided with project file for Visual C++ 2005.

   Note:

   WCELIBCEX is built to Static Library. For details, check README.txt file form the package.

3. Download GDAL 1.4.0 release or directly from CVS

   Go to `http://www.gdal.org/download.html` and download ZIP package with GDAL 1.4.0. You can also checkout sources directly from SVN.

   For this guidelines, I assume following directories structure:

   ```
   C:\dev\gdal-1.4.0
   C:\dev\wcelibcex-1.0
   ```

4. Projects configuration

   (a) Open gdalce_dll.sln project in Visual C++ 2005 IDE
   According to the paths presented in step 3, you should load following file:

   ```
   C:\dev\gdal-1.4.0\wince\msvc80\gdalce_dll\gdalce_dll.sln
   ```

   (b) Add WCELIBCEX project to gdalce_dll.sln solution
   Go to File -> Add -> Existing Project, navigage and open following file:

   ```
   C:\dev\wcelibcex-1.0\msvc80\wcelibcex_lib.vcproj
   ```

   (c) Configure path to WCELIBCEX source:

   - Go to View -> Property Manager to open property manager window
   - Expand tree below gdalce_dll -> Debug -> gdalce_common
   - Right-click on gdalce_common and select Properties
   - In Property Pages dialog, under Common Properties, go to User Macros
   - In macros list, double-click on macro named as WCELIBCEX_DIR
   - According paths assumed in step 3, change the macro value to:

```
C:\dev\wcelibcex-1.0\src
```

- Click OK to apply changes and close the dialog

(d) Configure *wcelibcex_lib.vcproj* as a dependency for *gdalce_dll.vcproj*

- Select gdalce_dll project in Solution Explorer
- Go to Project -> Project Dependencies
- In the 'Depends on:' pane, select checkbox next to wcelibcex_lib
- Click OK to apply and close

5. Ready to build GDAL for Windows CE

   Go to Build and select Build Solution

   After a few minutes, you should see GDAL DLL ready to use. For example, when Pocket PC 2003 SDK is used and Debug configuration requested, all output files are located under this path:

   ```
   C:\dev\gdal-1.4.0\wince\msvc80\gdalce_dll\Pocket PC 2003 (ARMV4)\Debug
   ```

   There, you will find following binaries:

   - **gdalce.dll** - dymamic-link library
   - **gdalce_i.lib** - import library

35.5.1 Enable PROJ.4 support

PROJ.4 support is optional.

In the CVS repository of PROJ.4, there are available project files for Visual C++ 2005 for Windows CE.

It is recommended to read *README.txt* file from *wince\msvc80* directory in PROJ.4 sources tree. There, you will find instructions how to build PROJ.4 without attaching its project to gdalce_dll.sln. Then you can just add proj.dll and proj_i.lib to linker settings of gdalce_dll.vcproj project.

Below, you can find instructions how to add projce_dll.vcproj project directly to gdalce_dll.sln and build everything together.

1. Go to http://proj.maptools.org and learn how to checkout PROJ.4 source from the CVS
2. Checkout sources to prefered location, for example:

   ```
   C:\dev\proj
   ```

3. Add projce_dll.vcproj project to gdalce_dll.sln solution

   Go to File -> Add -> Existing Project, navigage and open following file:

   ```
   C:\dev\proj\wince\msvc80\projce_dll\projce_dll.vcproj
   ```

4. Open Property Manager as described here, open Property Page for gdalce_common, and edit macro named as PROJ_DIR.

   Change value of the PROJ_DIR macro to:

   ```
   C:\dev\proj
   ```

   Don't close the Property Manager yet.

5. Configure path to WCELIBCEX source:
   - Go to View -> Property Manager to open property manager window
   - Expand tree below projce_dll -> Debug -> projce_common
   - Right-click on projce_common and select Properties
   - In Property Pages dialog, under Common Properties, go to User Macros
   - In macros list, double-click on macro named as WCELIBCEX_DIR
   - According paths assumed in step 3, change the macro value to:

     ```
     C:\dev\wcelibcex-1.0\src
     ```

   - Click OK to apply changes and close the dialog
6. Follow instructions explained here and add projce_dll.vcproj as a dependency for gdalce_dll.vcproj
7. Update proj_config.h file:

   Go to *C:\dev\proj\src* and rename *proj_config.h.wince* to *proj_config.h*.
8. Ready to build GDAL for Windows CE

   Go to Build and select Build Solution

   Similarly to explanation above in step 5 for GDAL, binaries for PROJ.4 for Windows CE can be found here:

   ```
   C:\dev\proj\wince\msvc80\projce_dll\Pocket PC 2003 (ARMV4)\Debug
   ```

   There, you can find following binaries:

   - **proj.dll** - dymamic-link library
   - **proj_i.lib** - import library

   Note:

   PROJ.4 binaries for Windows CE do not include 'ce' in names. This is due the fact GDAL uses fixed proj.dll name to find and link dynamically with PROJ.4 DLL.
9. After all, put proj.dll to the same directory on device where you copied gdalce.dll and your application which uses GDAL.

35.6 How can I help?

I'd like to encourage everyone interested in using GDAL on Windows CE devices to help in its development. Here is a list of what you can do as a contribution to the project:

- You can build GDAL for Windows CE and report problems if you will meet any
- You can try to build new OGR drivers
- You can test GDAL/OGR on different Windows CE devices
- You can write sample applications using GDAL/OGR and announce them on the GDAL mailing list
- If you have found a bug or something is not working on the Windows CE, please report it on the GDAL's Bugzilla

There is also *wince\TODO* file where you can find list of things we are going to do.

If you have any comments or questions, please sent them to the gdal-dev@lists.maptools.org mailing list or directly to me on mateusz@loskot.net

Chapter 36

Deprecated List

Page GDAL for Windows CE Following directories and projects are deprecated. **DON'T USE THEM!**