GDAL

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** (p. **??**) to the calling application for all supported formats. It also comes with a variety of useful **commandline utilities** (p. **??**) 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

GDAL Virtual Format Tutorial

2.1 Introduction

The VRT driver is a format driver for GDAL that allows a virtual GDAL dataset to be composed from other GDAL datasets with repositioning, and algorithms potentially applied as well as various kinds of metadata altered or added. VRT descriptions of datasets can be saved in an XML format normally given the extension .vrt.

An example of a simple .vrt file referring to a 512x512 dataset with one band loaded from utm.tif might look like this:

```
<VRTDataset rasterXSize="512" rasterYSize="512">
  <GeoTransform>440720.0, 60.0, 0.0, 3751320.0, 0.0, -60.0</GeoTransform>
  <VRTRasterBand dataType="Byte" band="1">
    <ColorInterp>Gray</ColorInterp>
    <SimpleSource>
      <SourceFilename relativeToVRT="1">utm.tif</SourceFilename>
      <SourceBand>1</SourceBand>
      <SrcRect xOff="0" yOff="0" xSize="512" ySize="512"/>
      <DstRect xOff="0" yOff="0" xSize="512" ySize="512"/>
    </SimpleSource>
  </VRTRasterBand>
</VRTDataset>
```

Many aspects of the VRT file are a direct XML encoding of the `GDAL Data Model` which should be reviewed for understanding of the semantics of various elements.

VRT files can be produced by translating to VRT format. The resulting file can then be edited to modify mappings, add metadata or other purposes. VRT files can also be produced programmatically by various means.

This tutorial will cover the .vrt file format (suitable for users editing .vrt files), and how .vrt files may be created and manipulated programmatically for developers.

2.2 .vrt Format

Virtual files stored on disk are kept in an XML format with the following elements.

- **VRTDataset**: This is the root element for the whole GDAL dataset. It must have the attributes rasterXSize and rasterYSize describing the width and height of the dataset in pixels. It may have SRS, GeoTransform, GCPList, Metadata, and VRTRasterBand subelements.

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

- **SRS**: This element contains the spatial reference system (coordinate system) in OGC WKT format. Note that this must be appropriately escaped for XML, so items like quotes will have the ampersand escape sequences substituted. As as well WKT, and valid input to the SetFromUserInput() method (such as well known GEOGCS names, and PROJ.4 format) is also allowed in the SRS element.

  ```
  <SRS>PROJCS[&quot;NAD27 / UTM zone 11N&quot;,GEOGCS[&quot;NAD27&quot;,DATUM[&quot;North_American_Da
  ```

- **GeoTransform**: This element contains a six value affine geotransformation for the dataset, mapping between pixel/line coordinates and georeferenced coordinates.

  ```
  <GeoTransform>440720.0,  60,  0.0,  3751320.0,  0.0, -60.0</GeoTransform>
  ```

- **Metadata**: This element contains a list of metadata name/value pairs associated with the VRTDataset as a whole, or a VRTRasterBand. It has <MDI> (metadata item) subelements which have a "key" attribute and the value as the data of the element.

```
<Metadata>
  <MDI key="md_key">Metadata value</MDI>
</Metadata>
```

- **VRTRasterBand**: This represents one band of a dataset. It will have a ataType attribute with the type of the pixel data associated with this band (use names Byte, UInt16, Int16, UInt32, Int32, Float32, Float64, CInt16, CInt32, CFloat32 or CFloat64) and the band this element represents (1 based). This element may have Metadata, ColorInterp, NoDataValue, ColorTable, and Description subelements as well as the various kinds of source elements such as SimpleSource. A raster band may have many "sources" indicating where the actual raster data should be fetched from, and how it should be mapped into the raster bands pixel space.

- **ColorInterp**: The data of this element should be the name of a color interpretation type. One of Gray, Palette, Red, Green, Blue, Alpha, Hue, Saturation, Lightness, Cyan, Magenta, Yellow, Black, or Unknown.

```
<ColorInterp>Gray</ColorInterp>:
```

- **NoDataValue**: If this element exists a raster band has a nodata value associated with, of the value given as data in the element.

```
<NoDataValue>-100.0</NoDataValue>
```

- **ColorTable**: This element is parent to a set of Entry elements defining the entries in a color table. Currently only RGBA color tables are supported with c1 being red, c2 being green, c3 being blue and c4 being alpha. The entries are ordered and will be assumed to start from color table entry 0.

```
<ColorTable>
  <Entry c1="0" c2="0" c3="0" c4="255"/>
  <Entry c1="145" c2="78" c3="224" c4="255"/>
</ColorTable>
```

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

```
<Description>Crop Classification Layer</Description>
```

- **UnitType**: This optional element contains the vertical units for elevation band data. One of "m" for meters or "ft" for feet. Default assumption is meters.

```
<UnitType>ft</UnitType>
```

- **Offset**: This optional element contains the offset that should be applied when computing "real" pixel values from scaled pixel values on a raster band. The default is 0.0.

```
<Offset>0.0</Offset>
```

- **Scale**: This optional element contains the scale that should be applied when computing "real" pixel values from scaled pixel values on a raster band. The default is 1.0.

```
<Scale>0.0</Scale>
```

- **CategoryNames**: This optional element contains a list of Category subelements with the names of the categories for classified raster band.

```
<CategoryNames>
  <Category>Missing</Category>
  <Category>Non-Crop</Category>
  <Category>Wheat</Category>
  <Category>Corn</Category>
  <Category>Soybeans</Category>
</CategoryNames>
```

- **SimpleSource**: The SimpleSource indicates that raster data should be read from a separate dataset, indicating the dataset, and band to be read from, and how the data should map into this bands raster space. The SimpleSource may have the SourceFilename, SourceBand, SrcRect, and DstRect subelements. The SrcRect element will indicate what rectangle on the indicated source file should be read, and the DstRect element indicates how that rectangle of source data should be mapped into the VRTRasterBands space.

  The relativeToVRT attribute on the SourceFilename indicates whether the filename should be interpreted as relative to the .vrt file (value is 1) or not relative to the .vrt file (value is 0). The default is 0.

  Some characteristics of the source band can be specified in the optional SourceProperties tag to enable the VRT driver to differ the opening of the source dataset until it really needs to read data from it. This is particularly useful when building VRTs with a big number of source datasets. The needed parameters are the raster dimensions, the size of the blocks and the data type. If the SourceProperties tag is not present, the source dataset will be opened at the same time as the VRT itself.

```
<SimpleSource>
  <SourceFilename relativeToVRT="1">utm.tif</SourceFilename>
  <SourceBand>1</SourceBand>
  <SourceProperties RasterXSize="512" RasterYSize="512" DataType="Byte" BlockXSize="128" BlockYSi
  <SrcRect xOff="0" yOff="0" xSize="512" ySize="512"/>
  <DstRect xOff="0" yOff="0" xSize="512" ySize="512"/>
</SimpleSource>
```

- **AveragedSource**: The AveragedSource is derived from the SimpleSource and shares the same properties except that it uses an averaging resampling instead of a nearest neighbour algorithm as in SimpleSource, when the size of the destination rectangle is not the same as the size of the source rectangle

- **ComplexSource**: The ComplexSource is derived from the SimpleSource (so it shares the SourceFilename, SourceBand, SrcRect and DestRect elements), but it provides support to rescale and offset the range of the source values. Certain regions of the source can be masked by specifying the NODATA value.

  The ComplexSource supports adding a custom lookup table to transform the source values to the destination. The LUT can be specified using the following form:

```
<LUT>[src value 1]:[dest value 1],[src value 2]:[dest value 2],...</LUT>
```

  The intermediary values are calculated using a linear interpolation between the bounding destination values of the corresponding range.

  The ComplexSource supports fetching a color component from a source raster band that has a color table. The ColorTableComponent value is the index of the color component to extract : 1 for the red band, 2 for the green band, 3 for the blue band or 4 for the alpha band.

  When transforming the source values the operations are executed in the following order:

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

```
<ComplexSource>
  <SourceFilename relativeToVRT="1">utm.tif</SourceFilename>
  <SourceBand>1</SourceBand>
  <ScaleOffset>0</ScaleOffset>
  <ScaleRatio>1</ScaleRatio>
  <ColorTableComponent>1</ColorTableComponent>
  <LUT>0:0,2345.12:64,56789.5:128,2364753.02:255</LUT>
  <NODATA>0</NODATA>
  <SrcRect xOff="0" yOff="0" xSize="512" ySize="512"/>
  <DstRect xOff="0" yOff="0" xSize="512" ySize="512"/>
</ComplexSource>
```

- **KernelFilteredSource**: This is a pixel source derived from the Simple Source (so it shares the SourceFilename, SourceBand, SrcRect and DestRect elements, but it also passes the data through a simple filtering kernel specified with the Kernel element. The Kernel element should have two child elements, Size and Coefs and optionally the boolean attribute normalized (defaults to false=0). The size must always be an odd number, and the Coefs must have Size $*$ Size entries separated by spaces.

```
<KernelFilteredSource>
  <SourceFilename>/debian/home/warmerda/openev/utm.tif</SourceFilename>
  <SourceBand>1</SourceBand>
  <Kernel normalized="1">
    <Size>3</Size>
    <Coefs>0.11111111 0.11111111 0.11111111 0.11111111 0.11111111 0.11111111 0.11111111 0.111111
  </Kernel>
</KernelFilteredSource>
```

2.3 .vrt Descriptions for Raw Files

So far we have described how to derive new virtual datasets from existing files supports by GDAL. However, it is also common to need to utilize raw binary raster files for which the regular layout of the data is known but for which no format specific driver exists. This can be accomplished by writing a .vrt file describing the raw file.

For example, the following .vrt describes a raw raster file containing floating point complex pixels in a file called *l2p3hhsso.img*. The image data starts from the first byte (ImageOffset=0). The byte offset between pixels is 8 (PixelOffset=8), the size of a CFloat32. The byte offset from the start of one line to the start of the next is 9376 bytes (LineOffset=9376) which is the width (1172) times the size of a pixel (8).

```
<VRTDataset rasterXSize="1172" rasterYSize="1864">
  <VRTRasterBand dataType="CFloat32" band="1" subClass="VRTRawRasterBand">
    <SourceFilename relativetoVRT="1">l2p3hhsso.img</SourceFilename>
    <ImageOffset>0</ImageOffset>
    <PixelOffset>8</PixelOffset>
    <LineOffset>9376</LineOffset>
    <ByteOrder>MSB</ByteOrder>
  </VRTRasterBand>
</VRTDataset>
```

Some things to note are that the VRTRasterBand has a subClass specifier of "VRTRawRasterBand". Also, the VRTRasterBand contains a number of previously unseen elements but no "source" information. VRTRawRasterBands may never have sources (ie. SimpleSource), but should contain the following elements in addition to all the normal "metadata" elements previously described which are still supported.

- **SourceFilename**: The name of the raw file containing the data for this band. The relativeToVRT attribute can be used to indicate if the SourceFilename is relative to the .vrt file (1) or not (0).
- **ImageOffset**: The offset in bytes to the beginning of the first pixel of data of this image band. Defaults to zero.
- **PixelOffset**: The offset in bytes from the beginning of one pixel and the next on the same line. In packed single band data this will be the size of the **dataType** in bytes.
- **LineOffset**: The offset in bytes from the beginning of one scanline of data and the next scanline of data. In packed single band data this will be PixelOffset $*$ rasterXSize.
- **ByteOrder**: Defines the byte order of the data on disk. Either LSB (Least Significant Byte first) such as the natural byte order on Intel x86 systems or MSB (Most Significant Byte first) such as the natural byte order on Motorola or Sparc systems. Defaults to being the local machine order.

A few other notes:

- The image data on disk is assumed to be of the same data type as the band **dataType** of the VRTRawRasterBand.
- All the non-source attributes of the VRTRasterBand are supported, including color tables, metadata, nodata values, and color interpretation.
- The VRTRawRasterBand supports in place update of the raster, whereas the source based VRTRasterBand is always read-only.
- The OpenEV tool includes a File menu option to input parameters describing a raw raster file in a GUI and create the corresponding .vrt file.
- Multiple bands in the one .vrt file can come from the same raw file. Just ensure that the ImageOffset, PixelOffset, and LineOffset definition for each band is appropriate for the pixels of that particular band.

Another example, in this case a 400x300 RGB pixel interleaved image.

```
<VRTDataset rasterXSize="400" rasterYSize="300">
  <VRTRasterBand dataType="Byte" band="1" subClass="VRTRawRasterBand">
    <ColorInterp>Red</ColorInterp>
    <SourceFilename relativetoVRT="1">rgb.raw</SourceFilename>
    <ImageOffset>0</ImageOffset>
    <PixelOffset>3</PixelOffset>
    <LineOffset>1200</LineOffset>
  </VRTRasterBand>
  <VRTRasterBand dataType="Byte" band="2" subClass="VRTRawRasterBand">
    <ColorInterp>Green</ColorInterp>
    <SourceFilename relativetoVRT="1">rgb.raw</SourceFilename>
    <ImageOffset>1</ImageOffset>
    <PixelOffset>3</PixelOffset>
    <LineOffset>1200</LineOffset>
  </VRTRasterBand>
  <VRTRasterBand dataType="Byte" band="3" subClass="VRTRawRasterBand">
    <ColorInterp>Blue</ColorInterp>
    <SourceFilename relativetoVRT="1">rgb.raw</SourceFilename>
    <ImageOffset>2</ImageOffset>
    <PixelOffset>3</PixelOffset>
    <LineOffset>1200</LineOffset>
  </VRTRasterBand>
</VRTDataset>
```

2.4 Programatic Creation of VRT Datasets

The VRT driver supports several methods of creating VRT datasets. As of GDAL 1.2.0 the **vrtdataset.h** (p. **??**) include file should be installed with the core GDAL include files, allowing direct access to the VRT classes. However, even without that most capabilities remain available through standard GDAL interfaces.

To create a VRT dataset that is a clone of an existing dataset use the CreateCopy() method. For example to clone utm.tif into a wrk.vrt file in C++ the following could be used:

```
GDALDriver *poDriver = (GDALDriver *) GDALGetDriverByName( "VRT" );
GDALDataset *poSrcDS, *poVRTDS;

poSrcDS = (GDALDataset *) GDALOpenShared( "utm.tif", GA_ReadOnly );

poVRTDS =
  poDriver->CreateCopy( "wrk.vrt", poSrcDS, FALSE, NULL, NULL, NULL );
delete poVRTDS;
delete poSrcDS;
```

To create a virtual copy of a dataset with some attributes added or changed such as metadata or coordinate system that are often hard to change on other formats, you might do the following. In this case, the virtual dataset is created "in memory" only by virtual of creating it with an empty filename, and then used as a modified source to pass to a CreateCopy() written out in TIFF format.

```
poVRTDS = poDriver->CreateCopy( "", poSrcDS, FALSE, NULL, NULL, NULL );

poVRTDS->SetMetadataItem( "SourceAgency", "United States Geological Survey");
poVRTDS->SetMetadataItem( "SourceDate", "July 21, 2003" );

poVRTDS->GetRasterBand( 1 )->SetNoDataValue( -999.0 );

GDALDriver *poTIFFDriver = (GDALDriver *) GDALGetDriverByName( "GTiff" );
GDALDataset *poTiffDS;

poTiffDS =
  poTIFFDriver->CreateCopy( "wrk.tif", poVRTDS, FALSE, NULL, NULL, NULL );
delete poTiffDS;
```

In this example a virtual dataset is created with the Create() method, and adding bands and sources programmatically, but still via the "generic" API. A special attribute of VRT datasets is that sources can be added to the bands by passing the XML describing the source into SetMetadata() on the special domain target "new_vrt_sources". The domain target "vrt_sources" may also be used, in which case any existing sources will be discarded before adding the new ones. In this example we construct a simple averaging filter source instead of using the simple source.

```
  // construct XML for simple 3x3 average filter kernel source.
  const char *pszFilterSourceXML  =
"<KernelFilteredSource>"
"  <SourceFilename>utm.tif</SourceFilename>1<SourceBand>1</SourceBand>"
"  <Kernel>"
"    <Size>3</Size>"
"    <Coefs>0.111 0.111 0.111 0.111 0.111 0.111 0.111 0.111 0.111</Coefs>"
"  </Kernel>"
"</KernelFilteredSource>";

  // Create the virtual dataset.
  poVRTDS = poDriver->Create( "", 512, 512, 1, GDT_Byte, NULL );
  poVRTDS->GetRasterBand(1)->SetMetadataItem("source_0",pszFilterSourceXML",
                                             "new_vrt_sources");
```

A more general form of this that will produce a 3x3 average filtered clone of any input datasource might look like the following. In this case we deliberately set the filtered datasource as in the "vrt_sources" domain to override the SimpleSource created by the CreateCopy() method. The fact that we used CreateCopy() ensures that all the other metadata, georeferencing and so forth is preserved from the source dataset ... the only thing we are changing is the data source for each band.

```
int   nBand;
GDALDriver *poDriver = (GDALDriver *) GDALGetDriverByName( "VRT" );
GDALDataset *poSrcDS, *poVRTDS;

poSrcDS = (GDALDataset *) GDALOpenShared( pszSourceFilename, GA_ReadOnly );

poVRTDS = poDriver->CreateCopy( "", poSrcDS, FALSE, NULL, NULL, NULL );

for( nBand = 1; nBand <= poVRTDS->GetRasterCount(); nBand++ )
{
    char szFilterSourceXML[10000];

    GDALRasterBand *poBand = poVRTDS->GetRasterBand( nBand );

    sprintf( szFilterSourceXML,
      "<KernelFilteredSource>"
      "  <SourceFilename>%s</SourceFilename>1<SourceBand>%d</SourceBand>"
      "  <Kernel>"
      "    <Size>3</Size>"
      "    <Coefs>0.111 0.111 0.111 0.111 0.111 0.111 0.111 0.111 0.111</Coefs>"
      "  </Kernel>"
      "</KernelFilteredSource>",
      pszSourceFilename, nBand );

    poBand->SetMetadataItem( "source_0", szFilterSourceXML, "vrt_sources" );
}
```

Using Derived Bands

A specialized type of band is a 'derived' band which derives its pixel information from its source bands. With this type of band you must also specify a pixel function, which has the responsibility of generating the output raster. Pixel functions are created by an application and then registered with GDAL using a unique key.

Using derived bands you can create VRT datasets that manipulate bands on the fly without having to create new band files on disk. For example, you might want to generate a band using four source bands from a nine band input dataset (x0, x3, x4, and x8):

```
band_value = sqrt((x3*x3+x4*x4)/(x0*x8));
```

You could write the pixel function to compute this value and then register it with GDAL with the name "MyFirstFunction". Then, the following VRT XML could be used to display this derived band:

```
<VRTDataset rasterXSize="1000" rasterYSize="1000">
  <VRTRasterBand dataType="Float32" band="1" subClass="VRTDerivedRasterBand">>
    <Description>Magnitude</Description>
    <PixelFunctionType>MyFirstFunction</PixelFunctionType>
    <SimpleSource>
      <SourceFilename relativeToVRT="1">nine_band.dat</SourceFilename>
      <SourceBand>1</SourceBand>
      <SrcRect xOff="0" yOff="0" xSize="1000" ySize="1000"/>
      <DstRect xOff="0" yOff="0" xSize="1000" ySize="1000"/>
    </SimpleSource>
    <SimpleSource>
```

```
      <SourceFilename relativeToVRT="1">nine_band.dat</SourceFilename>
      <SourceBand>4</SourceBand>
      <SrcRect xOff="0" yOff="0" xSize="1000" ySize="1000"/>
      <DstRect xOff="0" yOff="0" xSize="1000" ySize="1000"/>
    </SimpleSource>
    <SimpleSource>
      <SourceFilename relativeToVRT="1">nine_band.dat</SourceFilename>
      <SourceBand>5</SourceBand>
      <SrcRect xOff="0" yOff="0" xSize="1000" ySize="1000"/>
      <DstRect xOff="0" yOff="0" xSize="1000" ySize="1000"/>
    </SimpleSource>
    <SimpleSource>
      <SourceFilename relativeToVRT="1">nine_band.dat</SourceFilename>
      <SourceBand>9</SourceBand>
      <SrcRect xOff="0" yOff="0" xSize="1000" ySize="1000"/>
      <DstRect xOff="0" yOff="0" xSize="1000" ySize="1000"/>
    </SimpleSource>
  </VRTRasterBand>
</VRTDataset>
```

In addition to the subclass specification (VRTDerivedRasterBand) and the PixelFunctionType value, there is another new parameter that can come in handy: SourceTransferType. Typically the source rasters are obtained using the data type of the derived band. There might be times, however, when you want the pixel function to have access to higher resolution source data than the data type being generated. For example, you might have a derived band of type "Float", which takes a single source of type "CFloat32" or "CFloat64", and returns the imaginary portion. To accomplish this, set the SourceTransferType to "CFloat64". Otherwise the source would be converted to "Float" prior to calling the pixel function, and the imaginary portion would be lost.

```
<VRTDataset rasterXSize="1000" rasterYSize="1000">
  <VRTRasterBand dataType="Float32" band="1" subClass="VRTDerivedRasterBand">>
    <Description>Magnitude</Description>
    <PixelFunctionType>MyFirstFunction</PixelFunctionType>
    <SourceTransferType>"CFloat64"</SourceTransferType>
    ...
```

Writing Pixel Functions

To register this function with GDAL (prior to accessing any VRT datasets with derived bands that use this function), an application calls GDALAddDerivedBandPixelFunc with a key and a GDALDerivedPixelFunc:

```
GDALAddDerivedBandPixelFunc("MyFirstFunction", TestFunction);
```

A good time to do this is at the beginning of an application when the GDAL drivers are registered.

GDALDerivedPixelFunc is defined with a signature similar to IRasterIO:

Parameters:

papoSources A pointer to packed rasters; one per source. The datatype of all will be the same, specified in the eSrcType parameter.

nSources The number of source rasters.

pData The buffer into which the data should be read, or from which it should be written. This buffer must contain at least nBufXSize $*$ nBufYSize words of type eBufType. It is organized in left to right, top to bottom pixel order. Spacing is controlled by the nPixelSpace, and nLineSpace parameters.

nBufXSize The width of the buffer image into which the desired region is to be read, or from which it is to be written.

nBufYSize The height of the buffer image into which the desired region is to be read, or from which it is to be written.

eSrcType The type of the pixel values in the papoSources raster array.

eBufType The type of the pixel values that the pixel function must generate in the pData data buffer.

nPixelSpace The byte offset from the start of one pixel value in pData to the start of the next pixel value within a scanline. If defaulted (0) the size of the datatype eBufType is used.

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

Returns:

CE_Failure on failure, otherwise CE_None.

```
typedef CPLErr
(*GDALDerivedPixelFunc)(void **papoSources, int nSources, void *pData,
                        int nXSize, int nYSize,
                        GDALDataType eSrcType, GDALDataType eBufType,
                        int nPixelSpace, int nLineSpace);
```

The following is an implementation of the pixel function:

```
#include "gdal.h"

CPLErr TestFunction(void **papoSources, int nSources, void *pData,
                    int nXSize, int nYSize,
                    GDALDataType eSrcType, GDALDataType eBufType,
                    int nPixelSpace, int nLineSpace)
{

    int ii, iLine, iCol;
    double pix_val;
    double x0, x3, x4, x8;

    /* ---- Init ---- */
    if (nSources != 4) return CE_Failure;

    /* ---- Set pixels ---- */
    for( iLine = 0; iLine < nYSize; iLine++ ) {
        for( iCol = 0; iCol < nXSize; iCol++ ) {
            ii = iLine * nXSize + iCol;

            /* Source raster pixels may be obtained with SRCVAL macro */
            x0 = SRCVAL(papoSources[0], eSrcType, ii);
            x3 = SRCVAL(papoSources[1], eSrcType, ii);
            x4 = SRCVAL(papoSources[2], eSrcType, ii);
            x8 = SRCVAL(papoSources[3], eSrcType, ii);

            pix_val = sqrt((x3*x3+x4*x4)/(x0*x8));

            GDALCopyWords(&pix_val, GDT_Float64, 0,
                          ((GByte *)pData) + nLineSpace * iLine +
                          iCol * nPixelSpace, eBufType, nPixelSpace, 1);
        }
    }

    /* ---- Return success ---- */
    return CE_None;
}
```

2.5 Multi-threading issues

When using VRT datasets in a multi-threading environment, you should be careful to open the VRT dataset by the thread that will use it afterwards. The reason for that is that the VRT dataset uses GDALOpenShared when opening the underlying datasets. So, if you open twice the same VRT dataset by the same thread, both VRT datasets will share the same handles to the underlying datasets.

Chapter 3

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

3.1 Sponsorship

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

3.1.1 Silver Sponsors

3.1.2 Other Sponsors

- `MicroImages Inc.`

3.2 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).

3.3 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 4

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 5

Simple C Example: gdalinfo.c

```
/******************************************************************************
 * $Id: gdalinfo.c 15032 2008-07-25 22:12:06Z warmerdam $
 *
 * Project:  GDAL Utilities
 * Purpose:  Commandline application to list info about a file.
 * Author:   Frank Warmerdam, warmerdam@pobox.com
 *
 * ****************************************************************************
 * Copyright (c) 1998, Frank Warmerdam
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and associated documentation files (the "Software"),
 * to deal in the Software without restriction, including without limitation
 * the rights to use, copy, modify, merge, publish, distribute, sublicense,
 * and/or sell copies of the Software, and to permit persons to whom the
 * Software is furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included
 * in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
 * OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
 * THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
 * DEALINGS IN THE SOFTWARE.
 ******************************************************************************/

#include "gdal.h"
#include "gdal_alg.h"
#include "ogr_srs_api.h"
#include "cpl_string.h"
#include "cpl_conv.h"
#include "cpl_multiproc.h"

CPL_CVSID("$Id: gdalinfo.c 15032 2008-07-25 22:12:06Z warmerdam $");

static int
GDALInfoReportCorner( GDALDatasetH hDataset,
                      OGRCoordinateTransformationH hTransform,
                      const char * corner_name,
                      double x, double y );

/************************************************************************/
/*                               Usage()                                */
/************************************************************************/

void Usage()

{
    printf( "Usage: gdalinfo [--help-general] [-mm] [-stats] [-hist] [-nogcp] [-nomd]\n"
            "                [-noct] [-checksum] [-mdd domain]* datasetname\n" );
    exit( 1 );
}

/************************************************************************/
/*                                main()                                */
/************************************************************************/

int main( int argc, char ** argv )

{
    GDALDatasetH        hDataset;
    GDALRasterBandH     hBand;
    int                 i, iBand;
    double              adfGeoTransform[6];
```

```
    GDALDriverH         hDriver;
    char                **papszMetadata;
    int                 bComputeMinMax = FALSE, bSample = FALSE;
    int                 bShowGCPs = TRUE, bShowMetadata = TRUE ;
    int                 bStats = FALSE, bApproxStats = TRUE, iMDD;
    int                 bShowColorTable = TRUE, bComputeChecksum = FALSE;
    int                 bReportHistograms = FALSE;
    const char          *pszFilename = NULL;
    char              **papszExtraMDDomains = NULL, **papszFileList;
    const char  *pszProjection = NULL;
    OGRCoordinateTransformationH hTransform = NULL;

    /* Check that we are running against at least GDAL 1.5 */
    /* Note to developers : if we use newer API, please change the requirement */
    if (atoi(GDALVersionInfo("VERSION_NUM")) < 1500)
    {
        fprintf(stderr, "At least, GDAL >= 1.5.0 is required for this version of %s, "
                "which was compiled against GDAL %s\n", argv[0], GDAL_RELEASE_NAME);
        exit(1);
    }

    GDALAllRegister();

    argc = GDALGeneralCmdLineProcessor( argc, &argv, 0 );
    if( argc < 1 )
        exit( -argc );

/* ------------------------------------------------------------------- */
/*      Parse arguments.                                               */
/* ------------------------------------------------------------------- */
    for( i = 1; i < argc; i++ )
    {
        if( EQUAL(argv[i], "--utility_version") )
        {
            printf("%s was compiled against GDAL %s and is running against GDAL %s\n",
                   argv[0], GDAL_RELEASE_NAME, GDALVersionInfo("RELEASE_NAME"));
            return 0;
        }
        else if( EQUAL(argv[i], "-mm") )
            bComputeMinMax = TRUE;
        else if( EQUAL(argv[i], "-hist") )
            bReportHistograms = TRUE;
        else if( EQUAL(argv[i], "-stats") )
        {
            bStats = TRUE;
            bApproxStats = FALSE;
        }
        else if( EQUAL(argv[i], "-approx_stats") )
        {
            bStats = TRUE;
            bApproxStats = TRUE;
        }
        else if( EQUAL(argv[i], "-sample") )
            bSample = TRUE;
        else if( EQUAL(argv[i], "-checksum") )
            bComputeChecksum = TRUE;
        else if( EQUAL(argv[i], "-nogcp") )
            bShowGCPs = FALSE;
        else if( EQUAL(argv[i], "-nomd") )
            bShowMetadata = FALSE;
        else if( EQUAL(argv[i], "-noct") )
            bShowColorTable = FALSE;
        else if( EQUAL(argv[i], "-mdd") && i < argc-1 )
            papszExtraMDDomains = CSLAddString( papszExtraMDDomains,
                                                argv[++i] );
        else if( argv[i][0] == '-' )
            Usage();
```

```
        else if( pszFilename == NULL )
            pszFilename = argv[i];
        else
            Usage();
    }

    if( pszFilename == NULL )
        Usage();

/* -------------------------------------------------------------------- */
/*      Open dataset.                                                   */
/* -------------------------------------------------------------------- */
    hDataset = GDALOpen( pszFilename, GA_ReadOnly );

    if( hDataset == NULL )
    {
        fprintf( stderr,
                 "gdalinfo failed - unable to open '%s'.\n",
                 pszFilename );

        CSLDestroy( argv );

        GDALDumpOpenDatasets( stderr );

        GDALDestroyDriverManager();

        CPLDumpSharedList( NULL );

        exit( 1 );
    }

/* -------------------------------------------------------------------- */
/*      Report general info.                                            */
/* -------------------------------------------------------------------- */
    hDriver = GDALGetDatasetDriver( hDataset );
    printf( "Driver: %s/%s\n",
            GDALGetDriverShortName( hDriver ),
            GDALGetDriverLongName( hDriver ) );

    papszFileList = GDALGetFileList( hDataset );
    if( CSLCount(papszFileList) == 0 )
    {
        printf( "Files: none associated\n" );
    }
    else
    {
        printf( "Files: %s\n", papszFileList[0] );
        for( i = 1; papszFileList[i] != NULL; i++ )
            printf( "       %s\n", papszFileList[i] );
    }
    CSLDestroy( papszFileList );

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

/* -------------------------------------------------------------------- */
/*      Report projection.                                              */
/* -------------------------------------------------------------------- */
    if( GDALGetProjectionRef( hDataset ) != NULL )
    {
        OGRSpatialReferenceH  hSRS;
        char                 *pszProjection;

        pszProjection = (char *) GDALGetProjectionRef( hDataset );

        hSRS = OSRNewSpatialReference(NULL);
```

```
        if( OSRImportFromWkt( hSRS, &pszProjection ) == CE_None )
        {
            char        *pszPrettyWkt = NULL;

            OSRExportToPrettyWkt( hSRS, &pszPrettyWkt, FALSE );
            printf( "Coordinate System is:\n%s\n", pszPrettyWkt );
            CPLFree( pszPrettyWkt );
        }
        else
            printf( "Coordinate System is `%s'\n",
                    GDALGetProjectionRef( hDataset ) );

        OSRDestroySpatialReference( hSRS );
    }

/* -------------------------------------------------------------------- */
/*      Report Geotransform.                                            */
/* -------------------------------------------------------------------- */
    if( GDALGetGeoTransform( hDataset, adfGeoTransform ) == CE_None )
    {
        if( adfGeoTransform[2] == 0.0 && adfGeoTransform[4] == 0.0 )
        {
            printf( "Origin = (%.15f,%.15f)\n",
                    adfGeoTransform[0], adfGeoTransform[3] );

            printf( "Pixel Size = (%.15f,%.15f)\n",
                    adfGeoTransform[1], adfGeoTransform[5] );
        }
        else
            printf( "GeoTransform =\n"
                    "  %.16g, %.16g, %.16g\n"
                    "  %.16g, %.16g, %.16g\n",
                    adfGeoTransform[0],
                    adfGeoTransform[1],
                    adfGeoTransform[2],
                    adfGeoTransform[3],
                    adfGeoTransform[4],
                    adfGeoTransform[5] );
    }

/* -------------------------------------------------------------------- */
/*      Report GCPs.                                                    */
/* -------------------------------------------------------------------- */
    if( bShowGCPs && GDALGetGCPCount( hDataset ) > 0 )
    {
        printf( "GCP Projection = %s\n", GDALGetGCPProjection(hDataset) );
        for( i = 0; i < GDALGetGCPCount(hDataset); i++ )
        {
            const GDAL_GCP      *psGCP;

            psGCP = GDALGetGCPs( hDataset ) + i;

            printf( "GCP[%3d]: Id=%s, Info=%s\n"
                    "          (%.15g,%.15g) -> (%.15g,%.15g,%.15g)\n",
                    i, psGCP->pszId, psGCP->pszInfo,
                    psGCP->dfGCPPixel, psGCP->dfGCPLine,
                    psGCP->dfGCPX, psGCP->dfGCPY, psGCP->dfGCPZ );
        }
    }

/* -------------------------------------------------------------------- */
/*      Report metadata.                                                */
/* -------------------------------------------------------------------- */
    papszMetadata = GDALGetMetadata( hDataset, NULL );
    if( bShowMetadata && CSLCount(papszMetadata) > 0 )
    {
        printf( "Metadata:\n" );
```

```
        for( i = 0; papszMetadata[i] != NULL; i++ )
        {
            printf( "  %s\n", papszMetadata[i] );
        }
    }

    for( iMDD = 0; iMDD < CSLCount(papszExtraMDDomains); iMDD++ )
    {
        papszMetadata = GDALGetMetadata( hDataset, papszExtraMDDomains[iMDD] );
        if( bShowMetadata && CSLCount(papszMetadata) > 0 )
        {
            printf( "Metadata (%s):\n", papszExtraMDDomains[iMDD]);
            for( i = 0; papszMetadata[i] != NULL; i++ )
            {
                printf( "  %s\n", papszMetadata[i] );
            }
        }
    }

/* -------------------------------------------------------------------- */
/*      Report "IMAGE_STRUCTURE" metadata.                              */
/* -------------------------------------------------------------------- */
    papszMetadata = GDALGetMetadata( hDataset, "IMAGE_STRUCTURE" );
    if( bShowMetadata && CSLCount(papszMetadata) > 0 )
    {
        printf( "Image Structure Metadata:\n" );
        for( i = 0; papszMetadata[i] != NULL; i++ )
        {
            printf( "  %s\n", papszMetadata[i] );
        }
    }

/* -------------------------------------------------------------------- */
/*      Report subdatasets.                                             */
/* -------------------------------------------------------------------- */
    papszMetadata = GDALGetMetadata( hDataset, "SUBDATASETS" );
    if( CSLCount(papszMetadata) > 0 )
    {
        printf( "Subdatasets:\n" );
        for( i = 0; papszMetadata[i] != NULL; i++ )
        {
            printf( "  %s\n", papszMetadata[i] );
        }
    }

/* -------------------------------------------------------------------- */
/*      Report geolocation.                                             */
/* -------------------------------------------------------------------- */
    papszMetadata = GDALGetMetadata( hDataset, "GEOLOCATION" );
    if( CSLCount(papszMetadata) > 0 )
    {
        printf( "Geolocation:\n" );
        for( i = 0; papszMetadata[i] != NULL; i++ )
        {
            printf( "  %s\n", papszMetadata[i] );
        }
    }

/* -------------------------------------------------------------------- */
/*      Report RPCs                                                     */
/* -------------------------------------------------------------------- */
    papszMetadata = GDALGetMetadata( hDataset, "RPC" );
    if( CSLCount(papszMetadata) > 0 )
    {
        printf( "RPC Metadata:\n" );
        for( i = 0; papszMetadata[i] != NULL; i++ )
        {
```

```
            printf( "  %s\n", papszMetadata[i] );
        }
    }

/* -------------------------------------------------------------------- */
/*      Setup projected to lat/long transform if appropriate.           */
/* -------------------------------------------------------------------- */
    if( GDALGetGeoTransform( hDataset, adfGeoTransform ) == CE_None )
        pszProjection = GDALGetProjectionRef(hDataset);

    if( pszProjection != NULL && strlen(pszProjection) > 0 )
    {
        OGRSpatialReferenceH hProj, hLatLong = NULL;

        hProj = OSRNewSpatialReference( pszProjection );
        if( hProj != NULL )
            hLatLong = OSRCloneGeogCS( hProj );

        if( hLatLong != NULL )
        {
            CPLPushErrorHandler( CPLQuietErrorHandler );
            hTransform = OCTNewCoordinateTransformation( hProj, hLatLong );
            CPLPopErrorHandler();

            OSRDestroySpatialReference( hLatLong );
        }

        if( hProj != NULL )
            OSRDestroySpatialReference( hProj );
    }

/* -------------------------------------------------------------------- */
/*      Report corners.                                                 */
/* -------------------------------------------------------------------- */
    printf( "Corner Coordinates:\n" );
    GDALInfoReportCorner( hDataset, hTransform, "Upper Left",
                          0.0, 0.0 );
    GDALInfoReportCorner( hDataset, hTransform, "Lower Left",
                          0.0, GDALGetRasterYSize(hDataset));
    GDALInfoReportCorner( hDataset, hTransform, "Upper Right",
                          GDALGetRasterXSize(hDataset), 0.0 );
    GDALInfoReportCorner( hDataset, hTransform, "Lower Right",
                          GDALGetRasterXSize(hDataset),
                          GDALGetRasterYSize(hDataset) );
    GDALInfoReportCorner( hDataset, hTransform, "Center",
                          GDALGetRasterXSize(hDataset)/2.0,
                          GDALGetRasterYSize(hDataset)/2.0 );

    if( hTransform != NULL )
    {
        OCTDestroyCoordinateTransformation( hTransform );
        hTransform = NULL;
    }

/* ==================================================================== */
/*      Loop over bands.                                                */
/* ==================================================================== */
    for( iBand = 0; iBand < GDALGetRasterCount( hDataset ); iBand++ )
    {
        double      dfMin, dfMax, adfCMinMax[2], dfNoData;
        int         bGotMin, bGotMax, bGotNodata, bSuccess;
        int         nBlockXSize, nBlockYSize, nMaskFlags;
        double      dfMean, dfStdDev;
        GDALColorTableH hTable;
        CPLErr      eErr;

        hBand = GDALGetRasterBand( hDataset, iBand+1 );
```

```
if( bSample )
{
    float afSample[10000];
    int   nCount;

    nCount = GDALGetRandomRasterSample( hBand, 10000, afSample );
    printf( "Got %d samples.\n", nCount );
}

GDALGetBlockSize( hBand, &nBlockXSize, &nBlockYSize );
printf( "Band %d Block=%dx%d Type=%s, ColorInterp=%s\n", iBand+1,
        nBlockXSize, nBlockYSize,
        GDALGetDataTypeName(
            GDALGetRasterDataType(hBand)),
        GDALGetColorInterpretationName(
            GDALGetRasterColorInterpretation(hBand)) );

if( GDALGetDescription( hBand ) != NULL
    && strlen(GDALGetDescription( hBand )) > 0 )
    printf( "  Description = %s\n", GDALGetDescription(hBand) );

dfMin = GDALGetRasterMinimum( hBand, &bGotMin );
dfMax = GDALGetRasterMaximum( hBand, &bGotMax );
if( bGotMin || bGotMax || bComputeMinMax )
{
    printf( "  " );
    if( bGotMin )
        printf( "Min=%.3f ", dfMin );
    if( bGotMax )
        printf( "Max=%.3f ", dfMax );

    if( bComputeMinMax )
    {
        GDALComputeRasterMinMax( hBand, FALSE, adfCMinMax );
        printf( "  Computed Min/Max=%.3f,%.3f",
                adfCMinMax[0], adfCMinMax[1] );
    }

    printf( "\n" );
}

eErr = GDALGetRasterStatistics( hBand, bApproxStats, bStats,
                                &dfMin, &dfMax, &dfMean, &dfStdDev );
if( eErr == CE_None )
{
    printf( "  Minimum=%.3f, Maximum=%.3f, Mean=%.3f, StdDev=%.3f\n",
            dfMin, dfMax, dfMean, dfStdDev );
}

if( bReportHistograms )
{
    int nBucketCount, *panHistogram = NULL;

    eErr = GDALGetDefaultHistogram( hBand, &dfMin, &dfMax,
                                    &nBucketCount, &panHistogram,
                                    TRUE, GDALTermProgress, NULL );
    if( eErr == CE_None )
    {
        int iBucket;

        printf( "  %d buckets from %g to %g:\n  ",
                nBucketCount, dfMin, dfMax );
        for( iBucket = 0; iBucket < nBucketCount; iBucket++ )
            printf( "%d ", panHistogram[iBucket] );
        printf( "\n" );
        CPLFree( panHistogram );
```

```
    }
}

if ( bComputeChecksum)
{
    printf( "  Checksum=%d\n",
            GDALChecksumImage(hBand, 0, 0,
                              GDALGetRasterXSize(hDataset),
                              GDALGetRasterYSize(hDataset)));
}

dfNoData = GDALGetRasterNoDataValue( hBand, &bGotNodata );
if( bGotNodata )
{
    printf( "  NoData Value=%.18g\n", dfNoData );
}

if( GDALGetOverviewCount(hBand) > 0 )
{
    int         iOverview;

    printf( "  Overviews: " );
    for( iOverview = 0;
         iOverview < GDALGetOverviewCount(hBand);
         iOverview++ )
    {
        GDALRasterBandH hOverview;
        const char *pszResampling = NULL;

        if( iOverview != 0 )
            printf( ", " );

        hOverview = GDALGetOverview( hBand, iOverview );
        printf( "%dx%d",
                GDALGetRasterBandXSize( hOverview ),
                GDALGetRasterBandYSize( hOverview ) );

        pszResampling =
            GDALGetMetadataItem( hOverview, "RESAMPLING", "" );

        if( pszResampling != NULL
            && EQUALN(pszResampling,"AVERAGE_BIT2",12) )
            printf( "*" );
    }
    printf( "\n" );

    if ( bComputeChecksum)
    {
        printf( "  Overviews checksum: " );
        for( iOverview = 0;
            iOverview < GDALGetOverviewCount(hBand);
            iOverview++ )
        {
            GDALRasterBandH     hOverview;

            if( iOverview != 0 )
                printf( ", " );

            hOverview = GDALGetOverview( hBand, iOverview );
            printf( "%d",
                    GDALChecksumImage(hOverview, 0, 0,
                              GDALGetRasterBandXSize(hOverview),
                              GDALGetRasterBandYSize(hOverview)));
        }
        printf( "\n" );
    }
}
```

```
if( GDALHasArbitraryOverviews( hBand ) )
{
    printf( "  Overviews: arbitrary\n" );
}

nMaskFlags = GDALGetMaskFlags( hBand );
if( (nMaskFlags & (GMF_NODATA|GMF_ALL_VALID)) == 0 )
{
    GDALRasterBandH hMaskBand = GDALGetMaskBand(hBand) ;

    printf( "  Mask Flags: " );
    if( nMaskFlags & GMF_PER_DATASET )
        printf( "PER_DATASET " );
    if( nMaskFlags & GMF_ALPHA )
        printf( "ALPHA " );
    if( nMaskFlags & GMF_NODATA )
        printf( "NODATA " );
    if( nMaskFlags & GMF_ALL_VALID )
        printf( "ALL_VALID " );
    printf( "\n" );

    if( hMaskBand != NULL &&
        GDALGetOverviewCount(hMaskBand) > 0 )
    {
        int             iOverview;

        printf( "  Overviews of mask band: " );
        for( iOverview = 0;
             iOverview < GDALGetOverviewCount(hMaskBand);
             iOverview++ )
        {
            GDALRasterBandH     hOverview;

            if( iOverview != 0 )
                printf( ", " );

            hOverview = GDALGetOverview( hMaskBand, iOverview );
            printf( "%dx%d",
                    GDALGetRasterBandXSize( hOverview ),
                    GDALGetRasterBandYSize( hOverview ) );
        }
        printf( "\n" );
    }
}

if( strlen(GDALGetRasterUnitType(hBand)) > 0 )
{
    printf( "  Unit Type: %s\n", GDALGetRasterUnitType(hBand) );
}

if( GDALGetRasterCategoryNames(hBand) != NULL )
{
    char **papszCategories = GDALGetRasterCategoryNames(hBand);
    int i;

    printf( "  Categories:\n" );
    for( i = 0; papszCategories[i] != NULL; i++ )
        printf( "    %3d: %s\n", i, papszCategories[i] );
}

if( GDALGetRasterScale( hBand, &bSuccess ) != 1.0
    || GDALGetRasterOffset( hBand, &bSuccess ) != 0.0 )
    printf( "  Offset: %.15g,   Scale:%.15g\n",
            GDALGetRasterOffset( hBand, &bSuccess ),
            GDALGetRasterScale( hBand, &bSuccess ) );
```

```
    papszMetadata = GDALGetMetadata( hBand, NULL );
    if( bShowMetadata && CSLCount(papszMetadata) > 0 )
    {
        printf( "  Metadata:\n" );
        for( i = 0; papszMetadata[i] != NULL; i++ )
        {
            printf( "    %s\n", papszMetadata[i] );
        }
    }

    papszMetadata = GDALGetMetadata( hBand, "IMAGE_STRUCTURE" );
    if( bShowMetadata && CSLCount(papszMetadata) > 0 )
    {
        printf( "  Image Structure Metadata:\n" );
        for( i = 0; papszMetadata[i] != NULL; i++ )
        {
            printf( "    %s\n", papszMetadata[i] );
        }
    }

    if( GDALGetRasterColorInterpretation(hBand) == GCI_PaletteIndex
        && (hTable = GDALGetRasterColorTable( hBand )) != NULL )
    {
        int                i;

        printf( "  Color Table (%s with %d entries)\n",
                GDALGetPaletteInterpretationName(
                    GDALGetPaletteInterpretation( hTable )),
                GDALGetColorEntryCount( hTable ) );

        if (bShowColorTable)
        {
            for( i = 0; i < GDALGetColorEntryCount( hTable ); i++ )
            {
                GDALColorEntry      sEntry;

                GDALGetColorEntryAsRGB( hTable, i, &sEntry );
                printf( "  %3d: %d,%d,%d,%d\n",
                        i,
                        sEntry.c1,
                        sEntry.c2,
                        sEntry.c3,
                        sEntry.c4 );
            }
        }
    }

    if( bShowMetadata && GDALGetDefaultRAT( hBand ) != NULL )
    {
        GDALRasterAttributeTableH hRAT = GDALGetDefaultRAT( hBand );

        GDALRATDumpReadable( hRAT, NULL );
    }
}

GDALClose( hDataset );

CSLDestroy( papszExtraMDDomains );
CSLDestroy( argv );

GDALDumpOpenDatasets( stderr );

GDALDestroyDriverManager();

CPLDumpSharedList( NULL );
CPLCleanupTLS();
```

```
    exit( 0 );
}

/************************************************************************/
/*                        GDALInfoReportCorner()                        */
/************************************************************************/

static int
GDALInfoReportCorner( GDALDatasetH hDataset,
                      OGRCoordinateTransformationH hTransform,
                      const char * corner_name,
                      double x, double y )

{
    double      dfGeoX, dfGeoY;
    double      adfGeoTransform[6];

    printf( "%-11s ", corner_name );

/* ---------------------------------------------------------------- */
/*      Transform the point into georeferenced coordinates.         */
/* ---------------------------------------------------------------- */
    if( GDALGetGeoTransform( hDataset, adfGeoTransform ) == CE_None )
    {
        dfGeoX = adfGeoTransform[0] + adfGeoTransform[1] * x
            + adfGeoTransform[2] * y;
        dfGeoY = adfGeoTransform[3] + adfGeoTransform[4] * x
            + adfGeoTransform[5] * y;
    }

    else
    {
        printf( "(%7.1f,%7.1f)\n", x, y );
        return FALSE;
    }

/* ---------------------------------------------------------------- */
/*      Report the georeferenced coordinates.                       */
/* ---------------------------------------------------------------- */
    if( ABS(dfGeoX) < 181 && ABS(dfGeoY) < 91 )
    {
        printf( "(%12.7f,%12.7f) ", dfGeoX, dfGeoY );

    }
    else
    {
        printf( "(%12.3f,%12.3f) ", dfGeoX, dfGeoY );
    }

/* ---------------------------------------------------------------- */
/*      Transform to latlong and report.                            */
/* ---------------------------------------------------------------- */
    if( hTransform != NULL
        && OCTTransform(hTransform,1,&dfGeoX,&dfGeoY,NULL) )
    {

        printf( "(%s,", GDALDecToDMS( dfGeoX, "Long", 2 ) );
        printf( "%s)", GDALDecToDMS( dfGeoY, "Lat", 2 ) );
    }

    printf( "\n" );

    return TRUE;
}
```

Chapter 6

Standard Driver Registration: gdalallregister.cpp

```
/******************************************************************************
 * $Id: gdalallregister.cpp 16026 2009-01-01 18:47:30Z warmerdam $
 *
 * Project:  GDAL Core
 * Purpose:  Implementation of GDALAllRegister(), primary format registration.
 * Author:   Frank Warmerdam, warmerdam@pobox.com
 *
 ******************************************************************************
 * Copyright (c) 1998, Frank Warmerdam
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and associated documentation files (the "Software"),
 * to deal in the Software without restriction, including without limitation
 * the rights to use, copy, modify, merge, publish, distribute, sublicense,
 * and/or sell copies of the Software, and to permit persons to whom the
 * Software is furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included
 * in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
 * OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
 * THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
 * DEALINGS IN THE SOFTWARE.
 ******************************************************************************/

#include "gdal_priv.h"
#include "gdal_frmts.h"

CPL_CVSID("$Id: gdalallregister.cpp 16026 2009-01-01 18:47:30Z warmerdam $");

#ifdef notdef
// we may have a use for this some day
static char *szConfiguredFormats = "GDAL_FORMATS";
#endif

/************************************************************************/
/*                           GDALAllRegister()                          */
/*                                                                      */
/*      Register all identifiably supported formats.                    */
/************************************************************************/

void CPL_STDCALL GDALAllRegister()

{
    GetGDALDriverManager()->AutoLoadDrivers();

#ifdef FRMT_vrt
    GDALRegister_VRT();
#endif

#ifdef FRMT_gdb
    GDALRegister_GDB();
#endif

#ifdef FRMT_gtiff
    GDALRegister_GTiff();
#endif

#ifdef FRMT_nitf
    GDALRegister_NITF();
    GDALRegister_RPFTOC();
#endif
```

```
#ifdef FRMT_hfa
    GDALRegister_HFA();
#endif

#ifdef FRMT_ceos2
    GDALRegister_SAR_CEOS();
#endif

#ifdef FRMT_ceos
    GDALRegister_CEOS();
#endif

#ifdef FRMT_jaxapalsar
    GDALRegister_PALSARJaxa();
#endif

#ifdef FRMT_gff
    GDALRegister_GFF();
#endif

#ifdef FRMT_elas
    GDALRegister_ELAS();
#endif

#ifdef FRMT_aigrid
//    GDALRegister_AIGrid2();
    GDALRegister_AIGrid();
#endif

#ifdef FRMT_aaigrid
    GDALRegister_AAIGrid();
#endif

#ifdef FRMT_sdts
    GDALRegister_SDTS();
#endif

#ifdef FRMT_ogdi
    GDALRegister_OGDI();
#endif

#ifdef FRMT_dted
    GDALRegister_DTED();
#endif

#ifdef FRMT_png
    GDALRegister_PNG();
#endif

#ifdef FRMT_jpeg
    GDALRegister_JPEG();
#endif

#ifdef FRMT_mem
    GDALRegister_MEM();
#endif

#ifdef FRMT_jdem
    GDALRegister_JDEM();
#endif

#ifdef FRMT_gif
    GDALRegister_GIF();
    GDALRegister_BIGGIF();
#endif

#ifdef FRMT_envisat
```

```
    GDALRegister_Envisat();
#endif

#ifdef FRMT_fits
    GDALRegister_FITS();
#endif

#ifdef FRMT_bsb
    GDALRegister_BSB();
#endif

#ifdef FRMT_xpm
    GDALRegister_XPM();
#endif

#ifdef FRMT_bmp
    GDALRegister_BMP();
#endif

#ifdef FRMT_dimap
    GDALRegister_DIMAP();
#endif

#ifdef FRMT_airsar
    GDALRegister_AirSAR();
#endif

#ifdef FRMT_rs2
    GDALRegister_RS2();
#endif

#ifdef FRMT_pcidsk
    GDALRegister_PCIDSK();
#endif

#ifdef FRMT_pcraster
    GDALRegister_PCRaster();
#endif

#ifdef FRMT_ilwis
    GDALRegister_ILWIS();
#endif

#ifdef FRMT_sgi
    GDALRegister_SGI();
#endif

#ifdef FRMT_srtmhgt
    GDALRegister_SRTMHGT();
#endif

#ifdef FRMT_leveller
    GDALRegister_Leveller();
#endif

#ifdef FRMT_terragen
    GDALRegister_Terragen();
#endif

#ifdef FRMT_netcdf
    GDALRegister_GMT();
    GDALRegister_netCDF();
#endif

#ifdef FRMT_hdf4
    GDALRegister_HDF4();
    GDALRegister_HDF4Image();
```

```
#endif

#ifdef FRMT_pds
    GDALRegister_ISIS3();
    GDALRegister_ISIS2();
    GDALRegister_PDS();
#endif

#ifdef FRMT_ers
    GDALRegister_ERS();
#endif

#ifdef FRMT_jp2kak
// JPEG2000 support using Kakadu toolkit
    GDALRegister_JP2KAK();
#endif

#ifdef FRMT_ecw
    GDALRegister_ECW();
    GDALRegister_JP2ECW();
#endif

#ifdef FRMT_jpeg2000
// JPEG2000 support using JasPer toolkit
// This one should always be placed after other JasPer supported formats,
// such as BMP or PNM. In other case we will get bad side effects.
    GDALRegister_JPEG2000();
#endif

#ifdef FRMT_l1b
    GDALRegister_L1B();
#endif

#ifdef FRMT_fit
    GDALRegister_FIT();
#endif

#ifdef FRMT_grib
    GDALRegister_GRIB();
#endif

#ifdef FRMT_mrsid
    GDALRegister_MrSID();
#endif

#ifdef FRMT_rmf
    GDALRegister_RMF();
#endif

#ifdef FRMT_wcs
    GDALRegister_WCS();
#endif

#ifdef FRMT_wms
    GDALRegister_WMS();
#endif

#ifdef FRMT_sde
    GDALRegister_SDE();
#endif

#ifdef FRMT_msgn
    GDALRegister_MSGN();
#endif

#ifdef FRMT_msg
    GDALRegister_MSG();
```

```
#endif

#ifdef FRMT_idrisi
    GDALRegister_IDRISI();
#endif

#ifdef FRMT_ingr
    GDALRegister_INGR();
#endif

#ifdef FRMT_gsg
    GDALRegister_GSAG();
    GDALRegister_GSBG();
    GDALRegister_GS7BG();
#endif

#ifdef FRMT_cosar
    GDALRegister_COSAR();
#endif

#ifdef FRMT_tsx
    GDALRegister_TSX();
#endif

#ifdef FRMT_coasp
    GDALRegister_COASP();
#endif

#ifdef FRMT_tms
    GDALRegister_TMS();
#endif

/* -------------------------------------------------------------------- */
/*      Put raw formats at the end of the list. These drivers support   */
/*      various ASCII-header labeled formats, so the driver could be    */
/*      confused if you have files in some of above formats and such    */
/*      ASCII-header in the same directory.                             */
/* -------------------------------------------------------------------- */

#ifdef FRMT_raw
    GDALRegister_PNM();
    GDALRegister_DOQ1();
    GDALRegister_DOQ2();
    GDALRegister_ENVI();
    GDALRegister_EHdr();
    GDALRegister_GenBin();
    GDALRegister_PAux();
    GDALRegister_MFF();
    GDALRegister_HKV();
    GDALRegister_FujiBAS();
    GDALRegister_GSC();
    GDALRegister_FAST();
    GDALRegister_BT();
    GDALRegister_LAN();
    GDALRegister_CPG();
    GDALRegister_IDA();
    GDALRegister_NDF();
    GDALRegister_EIR();
    GDALRegister_DIPEx();
    GDALRegister_LCP();
#endif

/* -------------------------------------------------------------------- */
/*      Our test for the following is weak or expensive so we try       */
/*      them last.                                                      */
/* -------------------------------------------------------------------- */
```

```
#ifdef FRMT_rik
    GDALRegister_RIK();
#endif

#ifdef FRMT_usgsdem
    GDALRegister_USGSDEM();
#endif

#ifdef FRMT_gxf
    GDALRegister_GXF();
#endif

#ifdef FRMT_grass
    GDALRegister_GRASS();
#endif

#ifdef FRMT_dods
    GDALRegister_DODS();
#endif

#ifdef FRMT_wcs
    GDALRegister_HTTP();
#endif

#ifdef FRMT_hdf5
    GDALRegister_HDF5();
    GDALRegister_HDF5Image();
#endif

#ifdef FRMT_adrg
    GDALRegister_ADRG();
#endif

#ifdef FRMT_blx
    GDALRegister_BLX();
#endif

#ifdef FRMT_pgchip
    GDALRegister_PGCHIP();
#endif

#ifdef FRMT_georaster
    GDALRegister_GEOR();
#endif

/* ------------------------------------------------------------------- */
/*      Deregister any drivers explicitly marked as supressed by the   */
/*      GDAL_SKIP environment variable.                                */
/* ------------------------------------------------------------------- */
    GetGDALDriverManager()->AutoSkipDrivers();
}
```

Chapter 7

Sample Driver: jdemdataset.cpp

```
/******************************************************************************
 * $Id: jdemdataset.cpp 16257 2009-02-07 17:46:30Z rouault $
 *
 * Project:  JDEM Reader
 * Purpose:  All code for Japanese DEM Reader
 * Author:   Frank Warmerdam, warmerdam@pobox.com
 *
 ******************************************************************************
 * Copyright (c) 2000, Frank Warmerdam <warmerdam@pobox.com>
 *
 * Permission is hereby granted, free of charge, to any person obtaining a
 * copy of this software and associated documentation files (the "Software"),
 * to deal in the Software without restriction, including without limitation
 * the rights to use, copy, modify, merge, publish, distribute, sublicense,
 * and/or sell copies of the Software, and to permit persons to whom the
 * Software is furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included
 * in all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
 * OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
 * THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
 * DEALINGS IN THE SOFTWARE.
 ******************************************************************************/

#include "gdal_pam.h"

CPL_CVSID("$Id: jdemdataset.cpp 16257 2009-02-07 17:46:30Z rouault $");

CPL_C_START
void    GDALRegister_JDEM(void);
CPL_C_END

/************************************************************************/
/*                            JDEMGetField()                            */
/************************************************************************/

static int JDEMGetField( char *pszField, int nWidth )

{
    char        szWork[32];

    CPLAssert( nWidth < (int) sizeof(szWork) );

    strncpy( szWork, pszField, nWidth );
    szWork[nWidth] = '\0';

    return atoi(szWork);
}

/************************************************************************/
/*                           JDEMGetAngle()                             */
/************************************************************************/

static double JDEMGetAngle( char *pszField )

{
    int         nAngle = JDEMGetField( pszField, 7 );
    int         nDegree, nMin, nSec;

    // Note, this isn't very general purpose, but it would appear
    // from the field widths that angles are never negative.  Nice
    // to be a country in the "first quadrant".
```

```
    nDegree = nAngle / 10000;
    nMin = (nAngle / 100) % 100;
    nSec = nAngle % 100;

    return nDegree + nMin / 60.0 + nSec / 3600.0;
}

/************************************************************************/
/* ==================================================================== */
/*                              JDEMDataset                             */
/* ==================================================================== */
/************************************************************************/

class JDEMRasterBand;

class JDEMDataset : public GDALPamDataset
{
    friend class JDEMRasterBand;

    FILE        *fp;
    GByte       abyHeader[1012];

  public:
                ~JDEMDataset();

    static GDALDataset *Open( GDALOpenInfo * );

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

/************************************************************************/
/* ==================================================================== */
/*                            JDEMRasterBand                            */
/* ==================================================================== */
/************************************************************************/

class JDEMRasterBand : public GDALPamRasterBand
{
    friend class JDEMDataset;

    int          nRecordSize;
    char*        pszRecord;

  public:

                JDEMRasterBand( JDEMDataset *, int );
                ~JDEMRasterBand();

    virtual CPLErr IReadBlock( int, int, void * );
};


/************************************************************************/
/*                           JDEMRasterBand()                           */
/************************************************************************/

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

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

    eDataType = GDT_Float32;

    nBlockXSize = poDS->GetRasterXSize();
```

```
    nBlockYSize = 1;

    /* Cannot overflow as nBlockXSize <= 999 */
    nRecordSize = nBlockXSize*5 + 9 + 2;
    pszRecord = NULL;
}

/************************************************************************/
/*                           ~JDEMRasterBand()                          */
/************************************************************************/

JDEMRasterBand::~JDEMRasterBand()
{
    VSIFree(pszRecord);
}

/************************************************************************/
/*                              IReadBlock()                            */
/************************************************************************/

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

{
    JDEMDataset *poGDS = (JDEMDataset *) poDS;
    int         i;

    if (pszRecord == NULL)
    {
        if (nRecordSize < 0)
            return CE_Failure;

        pszRecord = (char *) VSIMalloc(nRecordSize);
        if (pszRecord == NULL)
        {
            CPLError(CE_Failure, CPLE_OutOfMemory,
                     "Cannot allocate scanline buffer");
            nRecordSize = -1;
            return CE_Failure;
        }
    }

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

    VSIFRead( pszRecord, 1, nRecordSize, poGDS->fp );

    if( !EQUALN((char *) poGDS->abyHeader,pszRecord,6) )
    {
        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 )
    {
        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] = (float)
            (JDEMGetField( pszRecord + 9 + 5 * i, 5) * 0.1);

    return CE_None;
```

```
}

/************************************************************************/
/* ==================================================================== */
/*                              JDEMDataset                             */
/* ==================================================================== */
/************************************************************************/

/************************************************************************/
/*                            ~JDEMDataset()                            */
/************************************************************************/

JDEMDataset::~JDEMDataset()

{
    FlushCache();
    if( fp != NULL )
        VSIFClose( fp );
}

/************************************************************************/
/*                          GetGeoTransform()                           */
/************************************************************************/

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;
}

/************************************************************************/
/*                          GetProjectionRef()                          */
/************************************************************************/

const char *JDEMDataset::GetProjectionRef()

{
    return( "GEOGCS[\"Tokyo\",DATUM[\"Tokyo\",SPHEROID[\"Bessel 1841\",6377397.155,299.1528128,AUTHORITY[\
}

/************************************************************************/
/*                                Open()                                */
/************************************************************************/

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 );
    if  (poDS->nRasterXSize <= 0 || poDS->nRasterYSize <= 0 )
    {
        CPLError( CE_Failure, CPLE_AppDefined,
                  "Invalid dimensions : %d x %d",
                  poDS->nRasterXSize, poDS->nRasterYSize);
        delete poDS;
        return NULL;
    }

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

/* ------------------------------------------------------------------- */
/*      Initialize any PAM information.                                */
/* ------------------------------------------------------------------- */
    poDS->SetDescription( poOpenInfo->pszFilename );
    poDS->TryLoadXML();

    return( poDS );
}

/************************************************************************/
/*                          GDALRegister_JDEM()                         */
/************************************************************************/

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 );
    }
}
```

Chapter 8

NEWS

Chapter 9

Building GDAL From Source

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

Chapter 10

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.

10.1 Dataset

A dataset (represented by the **GDALDataset** (p. **??**) 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.

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

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

10.1.2 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()** (p. **??**) 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).

10.1.3 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()** (p. **??**)). Each GCP (represented as the **GDAL_GCP** (p. **??**) class) contains the following:

```
typedef struct
{
    char *pszId;
    char *pszInfo;
    double  dfGCPPixel;
    double dfGCPLine;
    double dfGCPX;
    double dfGCPY;
    double dfGCPZ;
} GDAL_GCP (p. ??);
```

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.

10.1.4 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).

10.1.4.1 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()** (p. **??**) 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.

10.1.4.2 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.

10.1.4.3 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.

10.1.4.4 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.

10.2 Raster Band

A raster band is represented in GDAL with the **GDALRasterBand** (p. **??**) 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.

10.3 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 (p. ??);
```

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.

10.4 Overviews

A band may have zero or more overviews. Each overview is represented as a "free standing" **GDALRasterBand** (p. **??**). 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 11

GDAL Driver Implementation Tutorial

11.1 Overall Approach

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

11.2 Contents

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

11.3 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** (p. **??**) 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** (p. **??**)). 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** (p. **??**) 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 );
```

11.4 Implementing the RasterBand

Similar to the customized JDEMDataset class subclassed from **GDALDataset** (p. **??**), we also need to declare and implement a customized JDEMRasterBand derived from **GDALRasterBand** (p. **??**) 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** (p. **??**).

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** (p. **??**), and should generally be set in the band constructor.

- **poDS**: Pointer to the parent **GDALDataset** (p. **??**).
- **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** (p. **??**), 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.

11.5 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** (p. **??**). 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** (p. **??**) object when first called, and register it with the **GDALDriverManager** (p. **??**). 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)

11.6 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.

11.7 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.

11.8 Overviews

GDAL allows file formats to make pre-built overviews available to applications via the **GDALRasterBand::GetOverview()** (p. **??**) 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** (p. **??**), 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** (p. **??**) 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.

11.9 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** (p. **??**). 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.

11.9.1 CreateCopy

The **GDALDriver::CreateCopy()** (p. **??**) 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** (p. **??**) 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** (p. **??**) 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 );
}
```

11.9.2 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** (p. **??**).

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.

11.10 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** (p. **??**) 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.

11.11 Metadata, and Other Exotic Extensions

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

- **Metadata**: Name/value text values about a dataset or band. The **GDALMajorObject** (p. **??**) (base class for **GDALRasterBand** (p. **??**) and **GDALDataset** (p. **??**)) 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 12

gdal_polygonize

produces a polygon feature layer from a raster

12.1 SYNOPSIS

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

12.2 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()** (p. **??**) 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 13

gdal_proximity

produces a raster proximity map

13.1 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]
```

13.2 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 14

GDAL API Tutorial

14.1 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()** (p. **??**) function which attempts to register all known drivers, including those auto-loaded from .so files using **GDALDriverManager::AutoLoadDrivers()** (p. **??**). 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()** (p. **??**) 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()** (p. **??**) 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.

14.2 Getting Dataset Information

As described in the GDAL Data Model, a **GDALDataset** (p. **??**) 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],')'
```

14.3 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** (p. **??**) 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.'
```

14.4 Reading Raster Data

There are a few ways to read raster data, but the most common is via the **GDALRasterBand::RasterIO()** (p. **??**) 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.

14.5 Closing the Dataset

Please keep in mind that **GDALRasterBand** (p. **??**) 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()** (p. **??**) or using the delete operator on the **GDALDataset** (p. **??**). Either will result in proper cleanup, and flushing of any pending writes.

14.6 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()** (p. **??**) 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().

14.7 Using CreateCopy()

The **GDALDriver::CreateCopy()** (p. **??**) 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' ] )
```

14.8 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()** (p. **??**) 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 15

GDAL Utilities

The following utility programs are distributed with GDAL.

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

15.1 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.

15.2 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 16

gdalinfo

lists information about a raster dataset

16.1 SYNOPSIS

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

16.2 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.

16.3 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 17

gdal_translate

converts raster data between different formats

17.1 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
```

17.2 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.

17.3 EXAMPLE

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

Chapter 18

gdaladdo

builds or rebuilds overview images

18.1 SYNOPSIS

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

18.2 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)

18.3 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 19

gdaltindex

builds a shapefile as a raster tileindex

19.1 SYNOPSIS

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

19.2 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.

19.3 EXAMPLE

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

Chapter 20

gdalbuildvrt

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

20.1 SYNOPSIS

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

20.2 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** (p. **??**) 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/

20.3 EXAMPLE

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

Chapter 21

gdal_contour

builds vector contour lines from a raster elevation model

21.1 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>
```

21.2 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".

21.3 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 22

gdal_rasterize

burns vector polygons into a raster

22.1 SYNOPSIS

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

22.2 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.

22.3 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 23

rgb2pct.py

Convert a 24bit RGB image to 8bit paletted

23.1 SYNOPSIS

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

23.2 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 24

pct2rgb.py

Convert an 8bit paletted image to 24bit RGB

24.1 SYNOPSIS

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

24.2 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 25

gdaltransform

transforms coordinates

25.1 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]]
```

25.2 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()** (p. **??**).

-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.

25.3 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
```

25.4 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 26

nearblack

convert nearly black/white borders to black

26.1 SYNOPSIS

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

26.2 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 27

gdal_merge.py

mosaics a set of images

27.1 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
```

27.2 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 28

gdal2tiles.py

generates directory with TMS tiles, KMLs and simple web viewers

28.1 SYNOPSIS

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

28.2 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 29

gdal-config

determines various information about a GDAL installation

29.1 SYNOPSIS

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

29.2 DESCRIPTION

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

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

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

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

–version: Reports the GDAL version.

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

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

Chapter 30

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 31

gdal_grid

creates regular grid from the scattered data

31.1 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>
```

31.2 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** (p. **??**) and **DATA METRICS** (p. **??**) 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.

31.3 INTERPOLATION ALGORITHMS

There are number of interpolation algorithms to choose from.

31.3.1 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).

31.3.2 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.

31.3.3 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).

31.4 DATA METRICS

Besides the interpolation functionality **gdal_grid** (p. **??**) 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).

31.5 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** (p. **??**). All you need is create a virtual dataset header (VRT) for you CSV file and use it as input datasource for **gdal_grid** (p. **??**). 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.

31.6 EXAMPLE

The following would create raster TIFF file from VRT datasource described in **READING COMMA SEPARATED VALUES** (p. **??**) 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 32

GDAL Grid Tutorial

32.1 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 32.1: 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** (p. **??**)) or compute data metrics (see **Data Metrics Computation** (p. **??**)).

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

32.2 Interpolation of the Scattered Data

32.2.1 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** (p. **??**) ellipse".

In this method the weighting factor w is

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

See **GDALGridInverseDistanceToAPowerOptions** (p. **??**) for the list of **GDALGridCreate** (p. **??**) parameters and **invdist** (p. **??**) for the list of **gdal_grid** (p. **??**) options.

32.2.2 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** (p. **??**) 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** (p. **??**).

See **GDALGridMovingAverageOptions** (p. **??**) for the list of **GDALGridCreate** (p. **??**) parameters and **average** (p. **??**) for the list of **gdal_grid** (p. **??**) options.

32.2.3 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** (p. **??**) for the list of **GDALGridCreate** (p. **??**) parameters and **nearest** (p. **??**) for the list of **gdal_grid** (p. **??**) options.

32.3 Data Metrics Computation

All the metrics have the same set controlling options. See the **GDALGridDataMetricsOptions** (p. **??**).

32.3.1 Minimum Data Value

Minimum value found in grid node **search ellipse** (p. **??**). 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** (p. **??**) ellipse".

32.3.2 Maximum Data Value

Maximum value found in grid node **search ellipse** (p. **??**). 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** (p. **??**) ellipse".

32.3.3 Data Range

A difference between the minimum and maximum values found in grid node **search ellipse** (p. **??**). 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** (p. **??**) ellipse".

32.4 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 32.2: Search ellipse

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

Chapter 33

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.

33.1 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.

33.2 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.

33.3 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 34

GDAL VB6 Bindings Tutorial

34.1 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** (p. **??**), **GDALDataset** (p. **??**), **GDALRasterBand** (p. **??**), **GDALColorTable** (p. **??**), OGRSpatialReference and OGRCoordinateTransformation.

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

34.2 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** (p. **??**) class.
- GDALDataset.cls - The **GDALDataset** (p. **??**) class.
- GDALRasterBand.cls - The **GDALRasterBand** (p. **??**) class.
- GDALColorTable.cls - The **GDALColorTable** (p. **??**) 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.

34.3 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()** (p. **??**).

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

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

```
Dim MDValue As String

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

The **GDALDataset::GetGeoTransform()** (p. **??**) 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()** (p. **??**) 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 (p. **??**) objects have one or more raster bands associated with them. **GDALRasterBand** (p. **??**) objects can have metadata (accessed the same as on the **GDALDataset** (p. **??**)) 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** (p. **??**) has a DataType property which has the value returned by the C++ method **GDALRasterBand::GetRasterDataType()** (p. **??**). The returned value is an integer, but may be compared to the predefined constants **GDAL.GDT_Byte** (p. **??**), **GDAL.GDT_UInt16** (p. **??**), **GDAL.GDT_Int16** (p. **??**), **GDAL.GDT_UInt32** (p. **??**), **GDAL.GDT_Int32** (p. **??**), **GDAL.GDT_Float32** (p. **??**), **GDAL.GDT_Float64** (p. **??**), **GDAL.GDT_CInt16** (p. **??**), **GDAL.GDT_CInt32** (p. **??**), **GDAL.GDT_CFloat32** (p. **??**) and **GDAL.GDT_CFloat64** (p. **??**). 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** (p. **??**) objects associated with them. They are read with the **GDALRasterBand::GetColorTable()** (p. **??**) 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()** (p. **??**) 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** (p. **??**) 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()
```

34.4 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** (p. **??**) to do the creating. The **GDALDriver** (p. **??**) 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()** (p. **??**). 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()** (p. **??**) 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
```

34.5 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 35

GDAL Warp API Tutorial

35.1 Overview

The GDAL Warp API (declared in **gdalwarper.h** (p. **??**)) 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** (p. **??**) structure with the options to be utilized, instantiating a **GDALWarpOperation** (p. **??**) based on these options, and then invoking the **GDALWarpOperation::ChunkAndWarpImage()** (p. **??**) method to perform the warp options internally using the **GDALWarpKernel** (p. **??**) class.

35.2 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** (p. **??**) 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() (p. **??**) 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** (p. **??**) is instantiated using them, and the warp actually performed with **GDALWarpOperation::ChunkAndWarpImage()** (p. **??**). 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.

35.3 Other Warping Options

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

1. **GDALWarpOptions::dfWarpMemoryLimit** (p. **??**) - Set the maximum amount of memory to be used by the **GDALWarpOperation** (p. **??**) 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** (p. **??**) - 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** (p. **??**) - This is a string list of NAME=VALUE options passed to the warper. See the **GDALWarpOptions::papszWarpOptions** (p. **??**) 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.

35.4 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()** (p. **??**) 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.

35.5 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** (p. **??**) 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()** (p. **??**) 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** (p. **??**) 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()** (p. **??**) method instead of **GDALWarpOperation::ChunkAndWarpImage()** (p. **??**). 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.

35.6 Other Masking Options

The **GDALWarpOptions** (p. **??**) 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 36

GDAL for Windows CE

Overview (p. **??**)

Features (p. **??**)

Supported Platforms (p. **??**)

Content of 'wince' directory (p. **??**)

Building GDAL for Windows CE using Microsoft Visual C++ 2005 (p. **??**)

Enable PROJ.4 support (p. **??**)

wince_building_geos

How can I help? (p. **??**)

36.1 Overview

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

36.2 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

36.3 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

36.4 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

36.5 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

36.5.1 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.

36.6 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 37

gdalwarp

image reprojection and warping utility

37.1 SYNOPSIS

```
gdalwarp [--help-general] [--formats]
    [-s_srs srs_def] [-t_srs srs_def] [-to "NAME=VALUE"]
    [-order n] [-tps] [-rpc] [-geoloc] [-et err_threshold]
    [-te xmin ymin xmax ymax] [-tr xres yres] [-ts width height]
    [-wo "NAME=VALUE"] [-ot Byte/Int16/...] [-wt Byte/Int16]
    [-srcnodata "value [value...]"] [-dstnodata "value [value...]"] -dstalpha
    [-r resampling_method] [-wm memory_in_mb] [-multi] [-q]
    [-cutline datasource] [-cl layer] [-cwhere expression]
    [-csql statement] [-cblend dist_in_pixels]
    [-of format] [-co "NAME=VALUE"]*
    srcfile* dstfile
```

37.2 DESCRIPTION

The gdalwarp utility is an image mosaicing, reprojection and warping utility. The program can reproject to any supported projection, and can also apply GCPs stored with the image if the image is "raw" with control information.

-s_srs *srs def*: 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()** (p. **??**).

-order *n*: order of polynomial used for warping (1 to 3). The default is to select a polynomial order based on the number of GCPs.

-tps: Force use of thin plate spline transformer based on available GCPs.

-rpc: Force use of RPCs.

-geoloc: Force use of Geolocation Arrays.

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

-te *xmin ymin xmax ymax*: set georeferenced extents of output file to be created (in target SRS).

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

-ts *width height*: set output file size in pixels and lines

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

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

-wt *type*: Working pixel data type. The data type of pixels in the source image and destination image buffers.

-r *resampling_method*: Resampling method to use. Available methods are:

near: nearest neighbour resampling (default, fastest algorithm, worst interpolation quality).

bilinear: bilinear resampling.

cubic: cubic resampling.

cubicspline: cubic spline resampling.

lanczos: Lanczos windowed sinc resampling.

-srcnodata *value [value...]*: Set nodata masking values for input bands (different values can be supplied for each band). If more than one value is supplied all values should be quoted to keep them together as a single operating system argument. Masked values will not be used in interpolation. Use a value of `None` to ignore intrinsic nodata settings on the source dataset.

-dstnodata *value [value...]*: Set nodata values for output bands (different values can be supplied for each band). If more than one value is supplied all values should be quoted to keep them together as a single operating system argument. New files will be initialized to this value and if possible the nodata value will be recorded in the output file.

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

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

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

-q: Be quiet.

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

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

-cutline *datasource*: Enable use of a blend cutline from the name OGR support datasource.

-cl *layername*: Select the named layer from the cutline datasource.

-cwhere *expression*: Restrict desired cutline features based on attribute query.

-csql *query*: Select cutline features using an SQL query instead of from a layer with -cl.

-cblend *distance*: Set a blend distance to use to blend over cutlines (in pixels).

***srcfile*:** The source file name(s).

***dstfile*:** The destination file name.

Mosaicing into an existing output file is supported if the output file already exists. The spatial extent of the existing file will not be modified to accomodate new data, so you may have to remove it in that case.

Polygon cutlines may be used to restrict the the area of the destination file that may be updated, including blending. Cutline features must be in the georeferenced units of the destination file.

37.3 EXAMPLE

For instance, an eight bit spot scene stored in GeoTIFF with control points mapping the corners to lat/long could be warped to a UTM projection with a command like this:

```
gdalwarp -t_srs '+proj=utm +zone=11 +datum=WGS84' raw_spot.tif utm11.tif
```

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

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

Chapter 38

gdal_sieve

removes small raster polygons

38.1 SYNOPSIS

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

38.2 DESCRIPTION

The gdal_sieve.py script removes raster polygons smaller than a provided threshold size (in pixels) and replaces replaces them with the pixel value of the largest neighbour polygon. The result can be written back to the existing raster band, or copied into a new file.

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

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

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

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

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

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

srcfile The source raster file used to identify target pixels. Only the first band is used.

-nomask: Do not use the default validity mask for the input band (such as nodata, or alpha masks).

-mask *filename*: Use the first band of the specified file as a validity mask (zero is invalid, non-zero is valid).

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

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

Chapter 39

Deprecated List

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

Chapter 40

Class Index

40.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

_CPLList . . . ??
CPLODBCDriverInstaller . . . ??
CPLODBCSession . . . ??
CPLODBCStatement . . . ??
CPLXMLNode . . . ??
GDAL_GCP . . . ??
GDALColorEntry . . . ??
GDALColorTable . . . ??
GDALGridDataMetricsOptions . . . ??
GDALGridInverseDistanceToAPowerOptions . . . ??
GDALGridMovingAverageOptions . . . ??
GDALGridNearestNeighborOptions . . . ??
GDALMajorObject . . . ??
    GDALDataset . . . ??
        GDALPamDataset . . . ??
    GDALDriver . . . ??
    GDALDriverManager . . . ??
    GDALRasterBand . . . ??
GDALRasterAttributeTable . . . ??
GDALWarpKernel . . . ??
GDALWarpOperation . . . ??
GDALWarpOptions . . . ??

Chapter 41

Class Index

41.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

_CPLList . . . . . **??**
CPLODBCDriverInstaller . . . . . **??**
CPLODBCSession . . . . . **??**
CPLODBCStatement . . . . . **??**
CPLXMLNode . . . . . **??**
GDAL_GCP . . . . . **??**
GDALColorEntry . . . . . **??**
GDALColorTable . . . . . **??**
GDALDataset (A set of associated raster bands, usually from one file) . . . . . **??**
GDALDriver (Format specific driver) . . . . . **??**
GDALDriverManager . . . . . **??**
GDALGridDataMetricsOptions . . . . . **??**
GDALGridInverseDistanceToAPowerOptions . . . . . **??**
GDALGridMovingAverageOptions . . . . . **??**
GDALGridNearestNeighborOptions . . . . . **??**
GDALMajorObject (Object with metadata) . . . . . **??**
GDALPamDataset . . . . . **??**
GDALRasterAttributeTable (Raster Attribute Table container) . . . . . **??**
GDALRasterBand (A single raster band (or channel)) . . . . . **??**
GDALWarpKernel . . . . . **??**
GDALWarpOperation . . . . . **??**
GDALWarpOptions . . . . . **??**

Chapter 42

File Index

42.1 File List

Here is a list of all documented files with brief descriptions:

cpl_config.h . . . . . **??**
cpl_conv.h . . . . . **??**
cpl_csv.h . . . . . **??**
cpl_error.h . . . . . **??**
cpl_hash_set.h . . . . . **??**
cpl_http.h . . . . . **??**
cpl_list.h . . . . . **??**
cpl_minixml.h . . . . . **??**
cpl_minizip_ioapi.h . . . . . **??**
cpl_minizip_unzip.h . . . . . **??**
cpl_multiproc.h . . . . . **??**
cpl_odbc.h . . . . . **??**
cpl_port.h . . . . . **??**
cpl_quad_tree.h . . . . . **??**
cpl_string.h . . . . . **??**
cpl_vsi.h . . . . . **??**
cpl_vsi_virtual.h . . . . . **??**
cpl_win32ce_api.h . . . . . **??**
cpl_wince.h . . . . . **??**
cplkeywordparser.h . . . . . **??**
gdal.h . . . . . **??**
gdal_alg.h . . . . . **??**
gdal_alg_priv.h . . . . . **??**
gdal_frmts.h . . . . . **??**
gdal_pam.h . . . . . **??**
gdal_priv.h . . . . . **??**
gdal_proxy.h . . . . . **??**
gdal_rat.h . . . . . **??**
gdal_version.h . . . . . **??**
gdal_vrt.h . . . . . **??**
gdalgrid.h . . . . . **??**
gdaljp2metadata.h . . . . . **??**
gdalwarper.h . . . . . **??**

gvgcpfit.h . . . . . **??**
thinplatespline.h . . . . . **??**
vrtdataset.h . . . . . **??**

Chapter 43

Class Documentation

43.1 _CPLList Struct Reference

`#include <cpl_list.h>`

Public Attributes

- void * **pData**
- struct **_CPLList** * **psNext**

43.1.1 Detailed Description

List element structure.

43.1.2 Member Data Documentation

43.1.2.1 void* _CPLList::pData

Pointer to the data object. Should be allocated and frred by the caller.

Referenced by CPLHashSetDestroy(), CPLHashSetForeach(), CPLHashSetRemove(), CPLListAppend(), CPLListGetData(), and CPLListInsert().

43.1.2.2 struct _CPLList* _CPLList::psNext [read]

Pointer to the next element in list. NULL, if current element is the last one

Referenced by CPLHashSetDestroy(), CPLHashSetForeach(), CPLHashSetRemove(), CPLListAppend(), CPLListCount(), CPLListDestroy(), CPLListGet(), CPLListGetLast(), CPLListGetNext(), CPLListInsert(), and CPLListRemove().

The documentation for this struct was generated from the following file:

- **cpl_list.h**

43.2 CPLODBCDriverInstaller Class Reference

`#include <cpl_odbc.h>`

Public Member Functions

- int **InstallDriver** (const char *pszDriver, const char *pszPathIn, WORD fRequest=ODBC_-INSTALL_COMPLETE)
- int **RemoveDriver** (const char *pszDriverName, int fRemoveDSN=FALSE)
- int **GetUsageCount** () const
- const char * **GetPathOut** () const
- const char * **GetLastError** () const
- DWORD **GetLastErrorCode** () const

43.2.1 Detailed Description

A class providing functions to install or remove ODBC driver.

43.2.2 Member Function Documentation

43.2.2.1 int CPLODBCDriverInstaller::InstallDriver (const char * *pszDriver*, const char * *pszPathIn*, WORD *fRequest* = `ODBC_INSTALL_COMPLETE`)

Installs ODBC driver or updates definition of already installed driver. Interanally, it calls ODBC's SQLInstallDriverEx function.

Parameters:

pszDriver - The driver definition as a list of keyword-value pairs describing the driver (See ODBC API Reference).

pszPathIn - Full path of the target directory of the installation, or a null pointer (for unixODBC, NULL is passed).

fRequest - The fRequest argument must contain one of the following values: ODBC_INSTALL_-COMPLETE - (default) complete the installation request ODBC_INSTALL_INQUIRY - inquire about where a driver can be installed

Returns:

TRUE indicates success, FALSE if it fails.

43.2.2.2 int CPLODBCDriverInstaller::RemoveDriver (const char * *pszDriverName*, int *fRemoveDSN* = `FALSE`)

Removes or changes information about the driver from the Odbcinst.ini entry in the system information.

Parameters:

pszDriverName - The name of the driver as registered in the Odbcinst.ini key of the system information.

fRemoveDSN - TRUE: Remove DSNs associated with the driver specified in lpszDriver. FALSE: Do not remove DSNs associated with the driver specified in lpszDriver.

Returns:

The function returns TRUE if it is successful, FALSE if it fails. If no entry exists in the system information when this function is called, the function returns FALSE. In order to obtain usage count value, call GetUsageCount().

The documentation for this class was generated from the following files:

- **cpl_odbc.h**
- cpl_odbc.cpp

43.3 CPLODBCSession Class Reference

`#include <cpl_odbc.h>`

Public Member Functions

- int **EstablishSession** (const char *pszDSN, const char *pszUserid, const char *pszPassword)
- const char * **GetLastError** ()
- int **CloseSession** ()
- int **Failed** (int, HSTMT=NULL)
- HDBC **GetConnection** ()
- HENV **GetEnvironment** ()

43.3.1 Detailed Description

A class representing an ODBC database session.

Includes error collection services.

43.3.2 Member Function Documentation

43.3.2.1 int CPLODBCSession::EstablishSession (const char * *pszDSN*, const char * *pszUserid*, const char * *pszPassword*)

Connect to database and logon.

Parameters:

pszDSN The name of the DSN being used to connect. This is not optional.

pszUserid the userid to logon as, may be NULL if not not required, or provided by the DSN.

pszPassword the password to logon with. May be NULL if not required or provided by the DSN.

Returns:

TRUE on success or FALSE on failure. Call **GetLastError()** (p. **??**) to get details on failure.

References GetLastError().

43.3.2.2 const char * CPLODBCSession::GetLastError ()

Returns the last ODBC error message.

Returns:

pointer to an internal buffer with the error message in it. Do not free or alter. Will be an empty (but not NULL) string if there is no pending error info.

Referenced by EstablishSession(), and CPLODBCStatement::Fetch().

The documentation for this class was generated from the following files:

- **cpl_odbc.h**
- cpl_odbc.cpp

43.4 CPLODBCStatement Class Reference

`#include <cpl_odbc.h>`

Public Member Functions

- **CPLODBCStatement** (**CPLODBCSession** ∗)
- HSTMT **GetStatement** ()
- void **Clear** ()
- void **AppendEscaped** (const char ∗)
- void **Append** (const char ∗)
- void **Append** (int)
- void **Append** (double)
- int **Appendf** (const char ∗,...)
- const char ∗ **GetCommand** ()
- int **ExecuteSQL** (const char ∗=NULL)
- int **Fetch** (int nOrientation=SQL_FETCH_NEXT, int nOffset=0)
- void **ClearColumnData** ()
- int **GetColCount** ()
- const char ∗ **GetColName** (int)
- short **GetColType** (int)
- const char ∗ **GetColTypeName** (int)
- short **GetColSize** (int)
- short **GetColPrecision** (int)
- short **GetColNullable** (int)
- int **GetColId** (const char ∗)
- const char ∗ **GetColData** (int, const char ∗=NULL)
- const char ∗ **GetColData** (const char ∗, const char ∗=NULL)
- int **GetColDataLength** (int)
- int **GetColumns** (const char ∗pszTable, const char ∗pszCatalog=NULL, const char ∗pszSchema=NULL)
- int **GetPrimaryKeys** (const char ∗pszTable, const char ∗pszCatalog=NULL, const char ∗pszSchema=NULL)
- int **GetTables** (const char ∗pszCatalog=NULL, const char ∗pszSchema=NULL)
- void **DumpResult** (FILE ∗fp, int bShowSchema=FALSE)
- int **CollectResultsInfo** ()

Static Public Member Functions

- static CPLString **GetTypeName** (int)
- static SQLSMALLINT **GetTypeMapping** (SQLSMALLINT)

43.4.1 Detailed Description

Abstraction for statement, and resultset.

Includes methods for executing an SQL statement, and for accessing the resultset from that statement. Also provides for executing other ODBC requests that produce results sets such as SQLColumns() and SQLTables() requests.

43.4.2 Member Function Documentation

43.4.2.1 void CPLODBCStatement::Clear ()

Clear internal command text and result set definitions.

Referenced by ExecuteSQL().

43.4.2.2 void CPLODBCStatement::AppendEscaped (const char * *pszText*)

Append text to internal command.

The passed text is appended to the internal SQL command text after escaping any special characters so it can be used as a character string in an SQL statement.

Parameters:

pszText text to append.

References Append().

43.4.2.3 void CPLODBCStatement::Append (const char * *pszText*)

Append text to internal command.

The passed text is appended to the internal SQL command text.

Parameters:

pszText text to append.

Referenced by Append(), AppendEscaped(), Appendf(), and ExecuteSQL().

43.4.2.4 void CPLODBCStatement::Append (int *nValue*)

Append to internal command.

The passed value is formatted and appended to the internal SQL command text.

Parameters:

nValue value to append to the command.

References Append().

43.4.2.5 void CPLODBCStatement::Append (double *dfValue*)

Append to internal command.

The passed value is formatted and appended to the internal SQL command text.

Parameters:

dfValue value to append to the command.

References Append().

43.4.2.6 int CPLODBCStatement::Appendf (const char * *pszFormat*, ...)

Append to internal command.

The passed format is used to format other arguments and the result is appended to the internal command text. Long results may not be formatted properly, and should be appended with the direct **Append()** (p. **??**) methods.

Parameters:

pszFormat printf() style format string.

Returns:

FALSE if formatting fails dueto result being too large.

References Append().

43.4.2.7 int CPLODBCStatement::ExecuteSQL (const char * *pszStatement* = NULL)

Execute an SQL statement.

This method will execute the passed (or stored) SQL statement, and initialize information about the result-set if there is one. If a NULL statement is passed, the internal stored statement that has been previously set via **Append()** (p. **??**) or **Appendf()** (p. **??**) calls will be used.

Parameters:

pszStatement the SQL statement to execute, or NULL if the internally saved one should be used.

Returns:

TRUE on success or FALSE if there is an error. Error details can be fetched with OGRODBCSession::GetLastError().

References Append(), and Clear().

43.4.2.8 int CPLODBCStatement::Fetch (int *nOrientation* = SQL_FETCH_NEXT, int *nOffset* = 0)

Fetch a new record.

Requests the next row in the current resultset using the SQLFetchScroll() call. Note that many ODBC drivers only support the default forward fetching one record at a time. Only SQL_FETCH_NEXT (the default) should be considered reliable on all drivers.

Currently it isn't clear how to determine whether an error or a normal out of data condition has occured if **Fetch()** (p. **??**) fails.

Parameters:

nOrientation One of SQL_FETCH_NEXT, SQL_FETCH_LAST, SQL_FETCH_PRIOR, SQL_FETCH_ABSOLUTE, or SQL_FETCH_RELATIVE (default is SQL_FETCH_NEXT).

nOffset the offset (number of records), ignored for some orientations.

Returns:

TRUE if a new row is successfully fetched, or FALSE if not.

References CPLODBCSession::GetLastError(), and GetTypeMapping().

Referenced by DumpResult().

43.4.2.9 int CPLODBCStatement::GetColCount ()

Fetch the resultset column count.

Returns:

the column count, or zero if there is no resultset.

Referenced by DumpResult().

43.4.2.10 const char * CPLODBCStatement::GetColName (int *iCol*)

Fetch a column name.

Parameters:

iCol the zero based column index.

Returns:

NULL on failure (out of bounds column), or a pointer to an internal copy of the column name.

Referenced by DumpResult().

43.4.2.11 short CPLODBCStatement::GetColType (int *iCol*)

Fetch a column data type.

The return type code is a an ODBC SQL_ code, one of SQL_UNKNOWN_TYPE, SQL_CHAR, SQL_-NUMERIC, SQL_DECIMAL, SQL_INTEGER, SQL_SMALLINT, SQL_FLOAT, SQL_REAL, SQL_-DOUBLE, SQL_DATETIME, SQL_VARCHAR, SQL_TYPE_DATE, SQL_TYPE_TIME, SQL_TYPE_-TIMESTAMPT.

Parameters:

iCol the zero based column index.

Returns:

type code or -1 if the column is illegal.

Referenced by DumpResult().

43.4.2.12 const char * CPLODBCStatement::GetColTypeName (int *iCol*)

Fetch a column data type name.

Returns data source-dependent data type name; for example, "CHAR", "VARCHAR", "MONEY", "LONG VARBINAR", or "CHAR ( ) FOR BIT DATA".

Parameters:

iCol the zero based column index.

Returns:

NULL on failure (out of bounds column), or a pointer to an internal copy of the column dat type name.

43.4.2.13 short CPLODBCStatement::GetColSize (int *iCol*)

Fetch the column width.

Parameters:

iCol the zero based column index.

Returns:

column width, zero for unknown width columns.

Referenced by DumpResult().

43.4.2.14 short CPLODBCStatement::GetColPrecision (int *iCol*)

Fetch the column precision.

Parameters:

iCol the zero based column index.

Returns:

column precision, may be zero or the same as column size for columns to which it does not apply.

Referenced by DumpResult().

43.4.2.15 short CPLODBCStatement::GetColNullable (int *iCol*)

Fetch the column nullability.

Parameters:

iCol the zero based column index.

Returns:

TRUE if the column may contains or FALSE otherwise.

Referenced by DumpResult().

43.4.2.16 int CPLODBCStatement::GetColId (const char * *pszColName*)

Fetch column index.

Gets the column index corresponding with the passed name. The name comparisons are case insensitive.

Parameters:

pszColName the name to search for.

Returns:

the column index, or -1 if not found.

Referenced by GetColData().

43.4.2.17 const char * CPLODBCStatement::GetColData (int *iCol*, const char * *pszDefault* = NULL)

Fetch column data.

Fetches the data contents of the requested column for the currently loaded row. The result is returned as a string regardless of the column type. NULL is returned if an illegal column is given, or if the actual column is "NULL".

Parameters:

iCol the zero based column to fetch.

pszDefault the value to return if the column does not exist, or is NULL. Defaults to NULL.

Returns:

pointer to internal column data or NULL on failure.

Referenced by DumpResult(), and GetColData().

43.4.2.18 const char * CPLODBCStatement::GetColData (const char * *pszColName*, const char * *pszDefault* = NULL)

Fetch column data.

Fetches the data contents of the requested column for the currently loaded row. The result is returned as a string regardless of the column type. NULL is returned if an illegal column is given, or if the actual column is "NULL".

Parameters:

pszColName the name of the column requested.

pszDefault the value to return if the column does not exist, or is NULL. Defaults to NULL.

Returns:

pointer to internal column data or NULL on failure.

References GetColData(), and GetColId().

43.4.2.19 int CPLODBCStatement::GetColumns (const char * *pszTable*, const char * *pszCatalog* = NULL, const char * *pszSchema* = NULL)

Fetch column definitions for a table.

The SQLColumn() method is used to fetch the definitions for the columns of a table (or other queriable object such as a view). The column definitions are digested and used to populate the **CPLODBCStatement** (p. **??**) column definitions essentially as if a "SELECT * FROM tablename" had been done; however, no resultset will be available.

Parameters:

pszTable the name of the table to query information on. This should not be empty.

pszCatalog the catalog to find the table in, use NULL (the default) if no catalog is available.

pszSchema the schema to find the table in, use NULL (the default) if no schema is available.

Returns:

TRUE on success or FALSE on failure.

43.4.2.20 int CPLODBCStatement::GetPrimaryKeys (const char * *pszTable*, const char * *pszCatalog* = NULL, const char * *pszSchema* = NULL)

Fetch primary keys for a table.

The SQLPrimaryKeys() function is used to fetch a list of fields forming the primary key. The result is returned as a result set matching the SQLPrimaryKeys() function result set. The 4th column in the result set is the column name of the key, and if the result set contains only one record then that single field will be the complete primary key.

Parameters:

pszTable the name of the table to query information on. This should not be empty.

pszCatalog the catalog to find the table in, use NULL (the default) if no catalog is available.

pszSchema the schema to find the table in, use NULL (the default) if no schema is available.

Returns:

TRUE on success or FALSE on failure.

43.4.2.21 int CPLODBCStatement::GetTables (const char * *pszCatalog* = NULL, const char * *pszSchema* = NULL)

Fetch tables in database.

The SQLTables() function is used to fetch a list tables in the database. The result is returned as a result set matching the SQLTables() function result set. The 3rd column in the result set is the table name. Only tables of type "TABLE" are returned.

Parameters:

pszCatalog the catalog to find the table in, use NULL (the default) if no catalog is available.

pszSchema the schema to find the table in, use NULL (the default) if no schema is available.

Returns:

TRUE on success or FALSE on failure.

43.4.2.22 void CPLODBCStatement::DumpResult (FILE * *fp*, int *bShowSchema* = FALSE)

Dump resultset to file.

The contents of the current resultset are dumped in a simply formatted form to the provided file. If requested, the schema definition will be written first.

Parameters:

fp the file to write to. stdout or stderr are acceptable.

bShowSchema TRUE to force writing schema information for the rowset before the rowset data itself. Default is FALSE.

References Fetch(), GetColCount(), GetColData(), GetColName(), GetColNullable(), GetColPrecision(), GetColSize(), GetColType(), and GetTypeName().

43.4.2.23 CPLString CPLODBCStatement::GetTypeName (int *nTypeCode*) [static]

Get name for SQL column type.

Returns a string name for the indicated type code (as returned from **CPLODBCStatement::GetColType()** (p. **??**)).

Parameters:

nTypeCode the SQL_ code, such as SQL_CHAR.

Returns:

internal string, "UNKNOWN" if code not recognised.

Referenced by DumpResult().

43.4.2.24 SQLSMALLINT CPLODBCStatement::GetTypeMapping (SQLSMALLINT *nTypeCode*) [static]

Get appropriate C data type for SQL column type.

Returns a C data type code, corresponding to the indicated SQL data type code (as returned from **CPLODBCStatement::GetColType()** (p. **??**)).

Parameters:

nTypeCode the SQL_ code, such as SQL_CHAR.

Returns:

data type code. The valid code is always returned. If SQL code is not recognised, SQL_C_BINARY will be returned.

Referenced by Fetch().

The documentation for this class was generated from the following files:

- **cpl_odbc.h**
- cpl_odbc.cpp

43.5 CPLXMLNode Struct Reference

`#include <cpl_minixml.h>`

Public Attributes

- **CPLXMLNodeType eType**

  Node type.

- char * **pszValue**

  Node value.

- struct **CPLXMLNode** * **psNext**

  Next sibling.

- struct **CPLXMLNode** * **psChild**

  Child node.

43.5.1 Detailed Description

Document node structure.

This C structure is used to hold a single text fragment representing a component of the document when parsed. It should be allocated with the appropriate CPL function, and freed with **CPLDestroyXMLNode()** (p. **??**). The structure contents should not normally be altered by application code, but may be freely examined by application code.

Using the psChild and psNext pointers, a heirarchical tree structure for a document can be represented as a tree of **CPLXMLNode** (p. **??**) structures.

43.5.2 Member Data Documentation

43.5.2.1 CPLXMLNodeType CPLXMLNode::eType

Node type.

One of CXT_Element, CXT_Text, CXT_Attribute, CXT_Comment, or CXT_Literal.

Referenced by CPLAddXMLChild(), CPLCloneXMLTree(), CPLCreateXMLNode(), CPLGetXMLNode(), CPLGetXMLValue(), CPLSearchXMLNode(), CPLSetXMLValue(), and CPLStripXMLNamespace().

43.5.2.2 char* CPLXMLNode::pszValue

Node value.

For CXT_Element this is the name of the element, without the angle brackets. Note there is a single CXT_Element even when the document contains a start and end element tag. The node represents the pair. All text or other elements between the start and end tag will appear as children nodes of this CXT_Element node.

For CXT_Attribute the pszValue is the attribute name. The value of the attribute will be a CXT_Text child.

For CXT_Text this is the text itself (value of an attribute, or a text fragment between an element start and end tags.

For CXT_Literal it is all the literal text. Currently this is just used for !DOCTYPE lines, and the value would be the entire line.

For CXT_Comment the value is all the literal text within the comment, but not including the comment start/end indicators ("<–" and "- → ").

Referenced by CPLCloneXMLTree(), CPLCreateXMLNode(), CPLDestroyXMLNode(), CPLGetXMLNode(), CPLGetXMLValue(), CPLParseXMLString(), CPLSearchXMLNode(), CPLSetXMLValue(), CPLStripXMLNamespace(), and GDALValidateCreationOptions().

43.5.2.3 struct CPLXMLNode* CPLXMLNode::psNext [read]

Next sibling.

Pointer to next sibling, that is the next node appearing after this one that has the same parent as this node. NULL if this node is the last child of the parent element.

Referenced by CPLAddXMLChild(), CPLAddXMLSibling(), CPLCloneXMLTree(), CPLCreateXMLNode(), CPLDestroyXMLNode(), CPLGetXMLNode(), CPLGetXMLValue(), CPLRemoveXMLChild(), CPLSearchXMLNode(), CPLSerializeXMLTree(), CPLSetXMLValue(), CPLStripXMLNamespace(), and GDALValidateCreationOptions().

43.5.2.4 struct CPLXMLNode* CPLXMLNode::psChild [read]

Child node.

Pointer to first child node, if any. Only CXT_Element and CXT_Attribute nodes should have children. For CXT_Attribute it should be a single CXT_Text value node, while CXT_Element can have any kind of child. The full list of children for a node are identified by walking the psNext's starting with the psChild node.

Referenced by CPLAddXMLChild(), CPLCloneXMLTree(), CPLCreateXMLNode(), CPLDestroyXMLNode(), CPLGetXMLNode(), CPLGetXMLValue(), CPLRemoveXMLChild(), CPLSearchXMLNode(), CPLSetXMLValue(), CPLStripXMLNamespace(), and GDALValidateCreationOptions().

The documentation for this struct was generated from the following file:

- **cpl_minixml.h**

43.6 GDAL_GCP Struct Reference

`#include <gdal.h>`

Public Attributes

- char * **pszId**
- char * **pszInfo**
- double **dfGCPPixel**
- double **dfGCPLine**
- double **dfGCPX**
- double **dfGCPY**
- double **dfGCPZ**

43.6.1 Detailed Description

Ground Control Point

43.6.2 Member Data Documentation

43.6.2.1 char* GDAL_GCP::pszId

Unique identifier, often numeric

43.6.2.2 char* GDAL_GCP::pszInfo

Informational message or ""

43.6.2.3 double GDAL_GCP::dfGCPPixel

Pixel (x) location of GCP on raster

Referenced by GDALCreateGCPTransformer(), GDALCreateTPSTransformer(), and GDALGCPsToGeoTransform().

43.6.2.4 double GDAL_GCP::dfGCPLine

Line (y) location of GCP on raster

Referenced by GDALCreateGCPTransformer(), GDALCreateTPSTransformer(), and GDALGCPsToGeoTransform().

43.6.2.5 double GDAL_GCP::dfGCPX

X position of GCP in georeferenced space

Referenced by GDALCreateGCPTransformer(), GDALCreateTPSTransformer(), and GDALGCPsToGeoTransform().

43.6.2.6 double GDAL_GCP::dfGCPY

Y position of GCP in georeferenced space

Referenced by GDALCreateGCPTransformer(), GDALCreateTPSTransformer(), and GDALGCPsToGeoTransform().

43.6.2.7 double GDAL_GCP::dfGCPZ

Elevation of GCP, or zero if not known

The documentation for this struct was generated from the following file:

- **gdal.h**

43.7 GDALColorEntry Struct Reference

`#include <gdal.h>`

Public Attributes

- short **c1**
- short **c2**
- short **c3**
- short **c4**

43.7.1 Detailed Description

Color tuple

43.7.2 Member Data Documentation

43.7.2.1 short GDALColorEntry::c1

gray, red, cyan or hue

Referenced by GDALColorTable::CreateColorRamp(), GDALComputeMedianCutPCT(), GDALDitherRGB2PCT(), GDALRasterBand::GetIndexColorTranslationTo(), GDALRasterAttributeTable::InitializeFromColorTable(), GDALColorTable::SetColorEntry(), and GDALRasterAttributeTable::TranslateToColorTable().

43.7.2.2 short GDALColorEntry::c2

green, magenta, or lightness

Referenced by GDALColorTable::CreateColorRamp(), GDALComputeMedianCutPCT(), GDALDitherRGB2PCT(), GDALRasterBand::GetIndexColorTranslationTo(), GDALRasterAttributeTable::InitializeFromColorTable(), GDALColorTable::SetColorEntry(), and GDALRasterAttributeTable::TranslateToColorTable().

43.7.2.3 short GDALColorEntry::c3

blue, yellow, or saturation

Referenced by GDALColorTable::CreateColorRamp(), GDALComputeMedianCutPCT(), GDALDitherRGB2PCT(), GDALRasterBand::GetIndexColorTranslationTo(), GDALRasterAttributeTable::InitializeFromColorTable(), GDALColorTable::SetColorEntry(), and GDALRasterAttributeTable::TranslateToColorTable().

43.7.2.4 short GDALColorEntry::c4

alpha or blackband

Referenced by GDALColorTable::CreateColorRamp(), GDALComputeMedianCutPCT(), GDALRasterAttributeTable::InitializeFromColorTable(), GDALColorTable::SetColorEntry(), and GDALRasterAttributeTable::TranslateToColorTable().

The documentation for this struct was generated from the following file:

- **gdal.h**

43.8 GDALColorTable Class Reference

`#include <gdal_priv.h>`

Public Member Functions

- **GDALColorTable** (**GDALPaletteInterp**=GPI_RGB)
- ~**GDALColorTable** ()
- **GDALColorTable** * **Clone** () const
- **GDALPaletteInterp GetPaletteInterpretation** () const
- int **GetColorEntryCount** () const
- const **GDALColorEntry** * **GetColorEntry** (int) const
- int **GetColorEntryAsRGB** (int, **GDALColorEntry** *) const
- void **SetColorEntry** (int, const **GDALColorEntry** *)
- int **CreateColorRamp** (int, const **GDALColorEntry** *, int, const **GDALColorEntry** *)

43.8.1 Detailed Description

A color table / palette.

43.8.2 Constructor & Destructor Documentation

43.8.2.1 GDALColorTable::GDALColorTable (GDALPaletteInterp *eInterpIn* = GPI_RGB)

Construct a new color table.

This constructor is the same as the C GDALCreateColorTable() function.

Parameters:

eInterpIn the interpretation to be applied to **GDALColorEntry** (p. **??**) values.

Referenced by Clone().

43.8.2.2 GDALColorTable::~GDALColorTable ()

Destructor.

This descructor is the same as the C GDALDestroyColorTable() function.

43.8.3 Member Function Documentation

43.8.3.1 GDALColorTable * GDALColorTable::Clone () const

Make a copy of a color table.

This method is the same as the C function GDALCloneColorTable().

References GDALColorTable().

43.8.3.2 GDALPaletteInterp GDALColorTable::GetPaletteInterpretation () const

Fetch palette interpretation.

The returned value is used to interprete the values in the **GDALColorEntry** (p. **??**).

This method is the same as the C function GDALGetPaletteInterpretation().

Returns:

palette interpretation enumeration value, usually GPI_RGB.

Referenced by GDALRegenerateOverviews().

43.8.3.3 int GDALColorTable::GetColorEntryCount () const

Get number of color entries in table.

This method is the same as the function GDALGetColorEntryCount().

Returns:

the number of color entries.

Referenced by CreateColorRamp(), GDALRasterBand::GetIndexColorTranslationTo(), and GDALRasterAttributeTable::InitializeFromColorTable().

43.8.3.4 const GDALColorEntry * GDALColorTable::GetColorEntry (int *i*) const

Fetch a color entry from table.

This method is the same as the C function GDALGetColorEntry().

Parameters:

i entry offset from zero to **GetColorEntryCount()** (p. **??**)-1.

Returns:

pointer to internal color entry, or NULL if index is out of range.

Referenced by GDALRasterBand::GetIndexColorTranslationTo().

43.8.3.5 int GDALColorTable::GetColorEntryAsRGB (int *i*, GDALColorEntry * *poEntry*) const

Fetch a table entry in RGB format.

In theory this method should support translation of color palettes in non-RGB color spaces into RGB on the fly, but currently it only works on RGB color tables.

This method is the same as the C function GDALGetColorEntryAsRGB().

Parameters:

i entry offset from zero to **GetColorEntryCount()** (p. **??**)-1.

poEntry the existing **GDALColorEntry** (p. **??**) to be overrwritten with the RGB values.

Returns:

TRUE on success, or FALSE if the conversion isn't supported.

References GPI_RGB.

Referenced by GDALRasterAttributeTable::InitializeFromColorTable().

43.8.3.6 void GDALColorTable::SetColorEntry (int *i*, const GDALColorEntry * *poEntry*)

Set entry in color table.

Note that the passed in color entry is copied, and no internal reference to it is maintained. Also, the passed in entry must match the color interpretation of the table to which it is being assigned.

The table is grown as needed to hold the supplied offset.

This function is the same as the C function GDALSetColorEntry().

Parameters:

i entry offset from zero to **GetColorEntryCount()** (p. **??**)-1.

poEntry value to assign to table.

References GDALColorEntry::c1, GDALColorEntry::c2, GDALColorEntry::c3, and GDALColorEntry::c4.

Referenced by CreateColorRamp(), and GDALRasterAttributeTable::TranslateToColorTable().

43.8.3.7 int GDALColorTable::CreateColorRamp (int *nStartIndex*, const GDALColorEntry * *psStartColor*, int *nEndIndex*, const GDALColorEntry * *psEndColor*)

Create color ramp

Automatically creates a color ramp from one color entry to another. It can be called several times to create multiples ramps in the same color table.

This function is the same as the C function GDALCreateColorRamp().

Parameters:

nStartIndex index to start the ramp on the color table [0..255]

psStartColor a color entry value to start the ramp

nEndIndex index to end the ramp on the color table [0..255]

psEndColor a color entry value to end the ramp

Returns:

total number of entries, -1 to report error

References GDALColorEntry::c1, GDALColorEntry::c2, GDALColorEntry::c3, GDALColorEntry::c4, GetColorEntryCount(), and SetColorEntry().

The documentation for this class was generated from the following files:

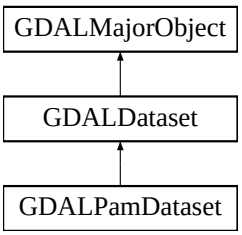
- gdal_priv.h
- gdalcolortable.cpp

43.9 GDALDataset Class Reference

A set of associated raster bands, usually from one file.

`#include <gdal_priv.h>`

Inheritance diagram for GDALDataset::



Public Member Functions

- virtual ~**GDALDataset** ()
- int **GetRasterXSize** (void)
- int **GetRasterYSize** (void)
- int **GetRasterCount** (void)
- **GDALRasterBand** * **GetRasterBand** (int)
- virtual void **FlushCache** (void)
- virtual const char * **GetProjectionRef** (void)
- virtual CPLErr **SetProjection** (const char *)
- virtual CPLErr **GetGeoTransform** (double *)
- virtual CPLErr **SetGeoTransform** (double *)
- virtual CPLErr **AddBand** (**GDALDataType** eType, char **papszOptions=NULL)
- virtual void * **GetInternalHandle** (const char *)
- virtual **GDALDriver** * **GetDriver** (void)
- virtual char ** **GetFileList** (void)
- virtual int **GetGCPCount** ()
- virtual const char * **GetGCPProjection** ()
- virtual const **GDAL_GCP** * **GetGCPs** ()
- virtual CPLErr **SetGCPs** (int nGCPCount, const **GDAL_GCP** *pasGCPList, const char *pszGCPProjection)
- virtual CPLErr **AdviseRead** (int nXOff, int nYOff, int nXSize, int nYSize, int nBufXSize, int nBufYSize, **GDALDataType** eDT, int nBandCount, int *panBandList, char **papszOptions)
- virtual CPLErr **CreateMaskBand** (int nFlags)
- CPLErr **RasterIO** (**GDALRWFlag**, int, int, int, int, void *, int, int, **GDALDataType**, int, int *, int, int, int)
- int **Reference** ()
- int **Dereference** ()
- **GDALAccess GetAccess** ()
- int **GetShared** ()
- void **MarkAsShared** ()
- CPLErr **BuildOverviews** (const char *, int, int *, int, int *, GDALProgressFunc, void *)

Static Public Member Functions

- static **GDALDataset** ** **GetOpenDatasets** (int *pnDatasetCount)

Protected Member Functions

- void **RasterInitialize** (int, int)
- void **SetBand** (int, **GDALRasterBand** *)
- virtual CPLErr **IBuildOverviews** (const char *, int, int *, int, int *, GDALProgressFunc, void *)
- virtual CPLErr **IRasterIO** (**GDALRWFlag**, int, int, int, int, void *, int, int, **GDALDataType**, int, int *, int, int, int)
- CPLErr **BlockBasedRasterIO** (**GDALRWFlag**, int, int, int, int, void *, int, int, **GDALDataType**, int, int *, int, int, int)
- void **BlockBasedFlushCache** ()

Protected Attributes

- **GDALDriver** * **poDriver**
- **GDALAccess eAccess**
- int **nRasterXSize**
- int **nRasterYSize**
- int **nBands**
- **GDALRasterBand** ** **papoBands**
- int **bForceCachedIO**
- int **nRefCount**
- int **bShared**
- GDALDefaultOverviews **oOvManager**

Friends

- class **GDALDriver**
- class **GDALDefaultOverviews**
- class **GDALProxyDataset**
- class **GDALRasterBand**
- GDALDatasetH **GDALOpen** (const char *, **GDALAccess**)
- GDALDatasetH **GDALOpenShared** (const char *, **GDALAccess**)

43.9.1 Detailed Description

A set of associated raster bands, usually from one file.

A dataset encapsulating one or more raster bands. Details are further discussed in the `GDAL Data Model`.

Use **GDALOpen()** (p. **??**) or **GDALOpenShared()** (p. **??**) to create a **GDALDataset** (p. **??**) for a named file, or **GDALDriver::Create()** (p. **??**) or **GDALDriver::CreateCopy()** (p. **??**) to create a new dataset.

43.9.2 Constructor & Destructor Documentation

43.9.2.1 GDALDataset::~GDALDataset () [virtual]

Destroy an open **GDALDataset** (p. **??**).

This is the accepted method of closing a GDAL dataset and deallocating all resources associated with it.

Equivelent of the C callable **GDALClose()** (p. **??**). Except that **GDALClose()** (p. **??**) first decrements the reference count, and then closes only if it has dropped to zero.

References GDALMajorObject::GetDescription().

43.9.3 Member Function Documentation

43.9.3.1 int GDALDataset::GetRasterXSize (void)

Fetch raster width in pixels.

Equivelent of the C function **GDALGetRasterXSize()** (p. **??**).

Returns:

the width in pixels of raster bands in this **GDALDataset** (p. **??**).

Referenced by GDALDatasetCopyWholeRaster(), and GDALRasterizeGeometries().

43.9.3.2 int GDALDataset::GetRasterYSize (void)

Fetch raster height in pixels.

Equivelent of the C function **GDALGetRasterYSize()** (p. **??**).

Returns:

the height in pixels of raster bands in this **GDALDataset** (p. **??**).

Referenced by GDALDatasetCopyWholeRaster(), and GDALRasterizeGeometries().

43.9.3.3 int GDALDataset::GetRasterCount (void)

Fetch the number of raster bands on this dataset.

Same as the C function **GDALGetRasterCount()** (p. **??**).

Returns:

the number of raster bands.

Referenced by BuildOverviews(), GDALDatasetCopyWholeRaster(), GDALGetRasterCount(), GDALRegenerateOverviews(), and RasterIO().

43.9.3.4 GDALRasterBand * GDALDataset::GetRasterBand (int *nBandId*)

Fetch a band object for a dataset.

Equivelent of the C function **GDALGetRasterBand()** (p. **??**).

Parameters:

nBandId the index number of the band to fetch, from 1 to **GetRasterCount()** (p. **??**).

Returns:

the height in pixels of raster bands in this **GDALDataset** (p. **??**).

Referenced by AdviseRead(), GDALDatasetCopyWholeRaster(), GDALGetRasterBand(), GDALRasterizeGeometries(), and RasterIO().

43.9.3.5 void GDALDataset::FlushCache (void) [virtual]

Flush all write cached data to disk.

Any raster (or other GDAL) data written via GDAL calls, but buffered internally will be written to disk.

This method is the same as the C function **GDALFlushCache()** (p. **??**).

Reimplemented in **GDALPamDataset** (p. **??**).

References GDALRasterBand::FlushCache().

Referenced by GDALPamDataset::FlushCache().

43.9.3.6 const char * GDALDataset::GetProjectionRef (void) [virtual]

Fetch the projection definition string for this dataset.

Same as the C function **GDALGetProjectionRef()** (p. **??**).

The returned string defines the projection coordinate system of the image in OpenGIS WKT format. It should be suitable for use with the OGRSpatialReference class.

When a projection definition is not available an empty (but not NULL) string is returned.

Returns:

a pointer to an internal projection reference string. It should not be altered, freed or expected to last for long.

See also:

`http://www.gdal.org/ogr/osr_tutorial.html`

Reimplemented in **GDALPamDataset** (p. **??**).

Referenced by GDALGetProjectionRef(), and GDALPamDataset::GetProjectionRef().

43.9.3.7 CPLErr GDALDataset::SetProjection (const char *) [virtual]

Set the projection reference string for this dataset.

The string should be in OGC WKT or PROJ.4 format. An error may occur because of incorrectly specified projection strings, because the dataset is not writable, or because the dataset does not support the indicated projection. Many formats do not support writing projections.

This method is the same as the C **GDALSetProjection()** (p. **??**) function.

Parameters:

pszProjection projection reference string.

Returns:

CE_Failure if an error occurs, otherwise CE_None.

Reimplemented in **GDALPamDataset** (p. **??**).

Referenced by GDALSetProjection(), and GDALPamDataset::SetProjection().

43.9.3.8 CPLErr GDALDataset::GetGeoTransform (double * *padfTransform*) [virtual]

Fetch the affine transformation coefficients.

Fetches the coefficients for transforming between pixel/line (P,L) raster space, and projection coordinates (Xp,Yp) space.

```
Xp = padfTransform[0] + P*padfTransform[1] + L*padfTransform[2];
Yp = padfTransform[3] + P*padfTransform[4] + L*padfTransform[5];
```

In a north up image, padfTransform[1] is the pixel width, and padfTransform[5] is the pixel height. The upper left corner of the upper left pixel is at position (padfTransform[0],padfTransform[3]).

The default transform is (0,1,0,0,0,1) and should be returned even when a CE_Failure error is returned, such as for formats that don't support transformation to projection coordinates.

NOTE: **GetGeoTransform()** (p. **??**) isn't expressive enough to handle the variety of OGC Grid Coverages pixel/line to projection transformation schemes. Eventually this method will be depreciated in favour of a more general scheme.

This method does the same thing as the C **GDALGetGeoTransform()** (p. **??**) function.

Parameters:

padfTransform an existing six double buffer into which the transformation will be placed.

Returns:

CE_None on success, or CE_Failure if no transform can be fetched.

Reimplemented in **GDALPamDataset** (p. **??**).

Referenced by GDALGetGeoTransform(), and GDALPamDataset::GetGeoTransform().

43.9.3.9 CPLErr GDALDataset::SetGeoTransform (double *) [virtual]

Set the affine transformation coefficients.

See **GetGeoTransform()** (p. **??**) for details on the meaning of the padfTransform coefficients.

This method does the same thing as the C **GDALSetGeoTransform()** (p. **??**) function.

Parameters:

padfTransform a six double buffer containing the transformation coefficients to be written with the dataset.

Returns:

CE_None on success, or CE_Failure if this transform cannot be written.

Reimplemented in **GDALPamDataset** (p. **??**).

Referenced by GDALSetGeoTransform(), and GDALPamDataset::SetGeoTransform().

43.9.3.10 CPLErr GDALDataset::AddBand (GDALDataType *eType*, char ** *papszOptions* = NULL) [virtual]

Add a band to a dataset.

This method will add a new band to the dataset if the underlying format supports this action. Most formats do not.

Note that the new **GDALRasterBand** (p. **??**) is not returned. It may be fetched after successful completion of the method by calling **GDALDataset::GetRasterBand** (p. **??**)(**GDALDataset::GetRasterCount()** (p. **??**)-1) as the newest band will always be the last band.

Parameters:

eType the data type of the pixels in the new band.

papszOptions a list of NAME=VALUE option strings. The supported options are format specific. NULL may be passed by default.

Returns:

CE_None on success or CE_Failure on failure.

43.9.3.11 void * GDALDataset::GetInternalHandle (const char *) [virtual]

Fetch a format specific internally meaningful handle.

This method is the same as the C **GDALGetInternalHandle()** (p. **??**) method.

Parameters:

pszHandleName the handle name desired. The meaningful names will be specific to the file format.

Returns:

the desired handle value, or NULL if not recognised/supported.

Referenced by GDALGetInternalHandle().

43.9.3.12 GDALDriver * GDALDataset::GetDriver (void) [virtual]

Fetch the driver to which this dataset relates.

This method is the same as the C **GDALGetDatasetDriver()** (p. **??**) function.

Returns:

the driver on which the dataset was created with **GDALOpen()** (p. **??**) or **GDALCreate()** (p. **??**).

43.9.3.13 char ** GDALDataset::GetFileList (void) [virtual]

Fetch files forming dataset.

Returns a list of files believed to be part of this dataset. If it returns an empty list of files it means there is believed to be no local file system files associated with the dataset (for instance a virtual dataset). The returned file list is owned by the caller and should be deallocated with **CSLDestroy()** (p. **??**).

The returned filenames will normally be relative or absolute paths depending on the path used to originally open the dataset.

This method is the same as the C **GDALGetFileList()** (p. **??**) function.

Returns:

NULL or a NULL terminated array of file names.

Reimplemented in **GDALPamDataset** (p. **??**).

References GDALMajorObject::GetDescription(), and VSIStatL().

Referenced by GDALPamDataset::GetFileList().

43.9.3.14 int GDALDataset::GetGCPCount () [virtual]

Get number of GCPs.

This method is the same as the C function **GDALGetGCPCount()** (p. **??**).

Returns:

number of GCPs for this dataset. Zero if there are none.

Reimplemented in **GDALPamDataset** (p. **??**).

Referenced by GDALPamDataset::GetGCPCount().

43.9.3.15 const char * GDALDataset::GetGCPProjection () [virtual]

Get output projection for GCPs.

This method is the same as the C function **GDALGetGCPProjection()** (p. **??**).

The projection string follows the normal rules from **GetProjectionRef()** (p. **??**).

Returns:

internal projection string or "" if there are no GCPs.

Reimplemented in **GDALPamDataset** (p. **??**).

Referenced by GDALPamDataset::GetGCPProjection().

43.9.3.16 const GDAL_GCP * GDALDataset::GetGCPs () [virtual]

Fetch GCPs.

This method is the same as the C function **GDALGetGCPs()** (p. **??**).

Returns:

pointer to internal GCP structure list. It should not be modified, and may change on the next GDAL call.

Reimplemented in **GDALPamDataset** (p. **??**).

Referenced by GDALPamDataset::GetGCPs().

43.9.3.17 CPLErr GDALDataset::SetGCPs (int *nGCPCount*, const GDAL_GCP * *pasGCPList*, const char * *pszGCPProjection*) [virtual]

Assign GCPs.

This method is the same as the C function **GDALSetGCPs()** (p. **??**).

This method assigns the passed set of GCPs to this dataset, as well as setting their coordinate system. Internally copies are made of the coordinate system and list of points, so the caller remains resposible for deallocating these arguments if appropriate.

Most formats do not support setting of GCPs, even foramts that can handle GCPs. These formats will return CE_Failure.

Parameters:

nGCPCount number of GCPs being assigned.

pasGCPList array of GCP structures being assign (nGCPCount in array).

pszGCPProjection the new OGC WKT coordinate system to assign for the GCP output coordinates. This parameter should be "" if no output coordinate system is known.

Returns:

CE_None on success, CE_Failure on failure (including if action is not supported for this format).

Reimplemented in **GDALPamDataset** (p. **??**).

Referenced by GDALPamDataset::SetGCPs().

43.9.3.18 CPLErr GDALDataset::AdviseRead (int *nXOff*, int *nYOff*, int *nXSize*, int *nYSize*, int *nBufXSize*, int *nBufYSize*, GDALDataType *eDT*, int *nBandCount*, int * *panBandMap*, char ** *papszOptions*) [virtual]

Advise driver of upcoming read requests.

Some GDAL drivers operate more efficiently if they know in advance what set of upcoming read requests will be made. The **AdviseRead()** (p. **??**) method allows an application to notify the driver of the region and bands of interest, and at what resolution the region will be read.

Many drivers just ignore the **AdviseRead()** (p. **??**) call, but it can dramatically accelerate access via some drivers.

Parameters:

nXOff The pixel offset to the top left corner of the region of the band to be accessed. This would be zero to start from the left side.

nYOff The line offset to the top left corner of the region of the band to be accessed. This would be zero to start from the top.

nXSize The width of the region of the band to be accessed in pixels.

nYSize The height of the region of the band to be accessed in lines.

nBufXSize the width of the buffer image into which the desired region is to be read, or from which it is to be written.

nBufYSize the height of the buffer image into which the desired region is to be read, or from which it is to be written.

eBufType the type of the pixel values in the pData data buffer. The pixel values will automatically be translated to/from the **GDALRasterBand** (p. **??**) data type as needed.

nBandCount the number of bands being read or written.

panBandMap the list of nBandCount band numbers being read/written. Note band numbers are 1 based. This may be NULL to select the first nBandCount bands.

papszOptions a list of name=value strings with special control options. Normally this is NULL.

Returns:

CE_Failure if the request is invalid and CE_None if it works or is ignored.

References GDALRasterBand::AdviseRead(), and GetRasterBand().

43.9.3.19 CPLErr GDALDataset::RasterIO (GDALRWFlag *eRWFlag*, int *nXOff*, int *nYOff*, int *nXSize*, int *nYSize*, void * *pData*, int *nBufXSize*, int *nBufYSize*, GDALDataType *eBufType*, int *nBandCount*, int * *panBandMap*, int *nPixelSpace*, int *nLineSpace*, int *nBandSpace*)

Read/write a region of image data from multiple bands.

This method allows reading a region of one or more GDALRasterBands from this dataset into a buffer, or writing data from a buffer into a region of the GDALRasterBands. It automatically takes care of data type translation if the data type (eBufType) of the buffer is different than that of the **GDALRasterBand** (p. **??**). The method also takes care of image decimation / replication if the buffer size (nBufXSize x nBufYSize) is different than the size of the region being accessed (nXSize x nYSize).

The nPixelSpace, nLineSpace and nBandSpace parameters allow reading into or writing from various organization of buffers.

For highest performance full resolution data access, read and write on "block boundaries" as returned by GetBlockSize(), or use the ReadBlock() and WriteBlock() methods.

This method is the same as the C **GDALDatasetRasterIO()** (p. **??**) function.

Parameters:

eRWFlag Either GF_Read to read a region of data, or GF_Write to write a region of data.

nXOff The pixel offset to the top left corner of the region of the band to be accessed. This would be zero to start from the left side.

nYOff The line offset to the top left corner of the region of the band to be accessed. This would be zero to start from the top.

nXSize The width of the region of the band to be accessed in pixels.

nYSize The height of the region of the band to be accessed in lines.

pData The buffer into which the data should be read, or from which it should be written. This buffer must contain at least nBufXSize ∗ nBufYSize ∗ nBandCount words of type eBufType. It is organized in left to right,top to bottom pixel order. Spacing is controlled by the nPixelSpace, and nLineSpace parameters.

nBufXSize the width of the buffer image into which the desired region is to be read, or from which it is to be written.

nBufYSize the height of the buffer image into which the desired region is to be read, or from which it is to be written.

eBufType the type of the pixel values in the pData data buffer. The pixel values will automatically be translated to/from the **GDALRasterBand** (p. **??**) data type as needed.

nBandCount the number of bands being read or written.

panBandMap the list of nBandCount band numbers being read/written. Note band numbers are 1 based. This may be NULL to select the first nBandCount bands.

nPixelSpace The byte offset from the start of one pixel value in pData to the start of the next pixel value within a scanline. If defaulted (0) the size of the datatype eBufType is used.

nLineSpace The byte offset from the start of one scanline in pData to the start of the next. If defaulted the size of the datatype eBufType ∗ nBufXSize is used.

nBandSpace the byte offset from the start of one bands data to the start of the next. If defaulted (zero) the value will be nLineSpace ∗ nBufYSize implying band sequential organization of the data buffer.

Returns:

CE_Failure if the access fails, otherwise CE_None.

References GDALGetDataTypeSize(), GetRasterBand(), GetRasterCount(), GF_Read, and GF_Write.

Referenced by GDALDatasetCopyWholeRaster(), GDALDatasetRasterIO(), and GDALRasterizeGeometries().

43.9.3.20 int GDALDataset::Reference ()

Add one to dataset reference count.

The reference is one after instantiation.

This method is the same as the C **GDALReferenceDataset()** (p. **??**) function.

Returns:

the post-increment reference count.

43.9.3.21 int GDALDataset::Dereference ()

Subtract one from dataset reference count.

The reference is one after instantiation. Generally when the reference count has dropped to zero the dataset may be safely deleted (closed).

This method is the same as the C **GDALDereferenceDataset()** (p. **??**) function.

Returns:

the post-decrement reference count.

Referenced by GDALClose().

43.9.3.22 int GDALDataset::GetShared ()

Returns shared flag.

Returns:

TRUE if the **GDALDataset** (p. **??**) is available for sharing, or FALSE if not.

Referenced by GDALClose().

43.9.3.23 void GDALDataset::MarkAsShared ()

Mark this dataset as available for sharing.

References GDALMajorObject::GetDescription().

Referenced by GDALOpenShared().

43.9.3.24 GDALDataset ** GDALDataset::GetOpenDatasets (int * *pnCount*) [static]

Fetch all open GDAL dataset handles.

This method is the same as the C function **GDALGetOpenDatasets()** (p. **??**).

NOTE: This method is not thread safe. The returned list may changed at any time.

Parameters:

pnCount integer into which to place the count of dataset pointers being returned.

Returns:

a pointer to an array of dataset handles.

Referenced by GDALGetOpenDatasets().

43.9.3.25 CPLErr GDALDataset::BuildOverviews (const char * *pszResampling*, int *nOverviews*, int * *panOverviewList*, int *nListBands*, int * *panBandList*, GDALProgressFunc *pfnProgress*, void * *pProgressData*)

Build raster overview(s)

If the operation is unsupported for the indicated dataset, then CE_Failure is returned, and **CPLGetLastErrorNo()** (p. **??**) will return CPLE_NotSupported.

This method is the same as the C function **GDALBuildOverviews()** (p. **??**).

Parameters:

pszResampling one of "NEAREST", "GAUSS", "AVERAGE", "AVERAGE_MAGPHASE" or "NONE" controlling the downsampling method applied.

nOverviews number of overviews to build.

panOverviewList the list of overview decimation factors to build.

nBand number of bands to build overviews for in panBandList. Build for all bands if this is 0.

panBandList list of band numbers.

pfnProgress a function to call to report progress, or NULL.

pProgressData application data to pass to the progress function.

Returns:

CE_None on success or CE_Failure if the operation doesn't work.

For example, to build overview level 2, 4 and 8 on all bands the following call could be made:

```
int       anOverviewList[3] = { 2, 4, 8 };

poDataset->BuildOverviews( "NEAREST", 3, anOverviewList, 0, NULL,
                           GDALDummyProgress, NULL );
```

See also:

GDALRegenerateOverviews() (p. **??**)

References GetRasterCount().

43.9.4 Friends And Related Function Documentation

43.9.4.1 GDALDatasetH GDALOpen (const char * *pszFilename*, GDALAccess *eAccess*) [friend]

Open a raster file as a **GDALDataset** (p. **??**).

This function will try to open the passed file, or virtual dataset name by invoking the Open method of each registered **GDALDriver** (p. **??**) in turn. The first successful open will result in a returned dataset. If all drivers fail then NULL is returned.

Several recommandations :

- If you open a dataset object with GA_Update access, it is not recommanded to open a new dataset on the same underlying file.
- The returned dataset should only be accessed by one thread at a time. If you want to use it from different threads, you must add all necessary code (mutexes, etc.) to avoid concurrent use of the object. (Some drivers, such as GeoTIFF, maintain internal state variables that are updated each time a new block is read, thus preventing concurrent use.)

See also:

GDALOpenShared() (p. **??**)

Parameters:

pszFilename the name of the file to access. In the case of exotic drivers this may not refer to a physical file, but instead contain information for the driver on how to access a dataset.

eAccess the desired access, either GA_Update or GA_ReadOnly. Many drivers support only read only access.

Returns:

A GDALDatasetH handle or NULL on failure. For C++ applications this handle can be cast to a **GDALDataset** (p. **??**) *.

Referenced by GDALOpenShared().

43.9.4.2 GDALDatasetH GDALOpenShared (const char * *pszFilename*, GDALAccess *eAccess*) [friend]

Open a raster file as a **GDALDataset** (p. **??**).

This function works the same as **GDALOpen()** (p. **??**), but allows the sharing of **GDALDataset** (p. **??**) handles for a dataset with other callers to **GDALOpenShared()** (p. **??**).

In particular, **GDALOpenShared()** (p. **??**) will first consult it's list of currently open and shared GDALDataset's, and if the **GetDescription()** (p. **??**) name for one exactly matches the pszFilename passed to **GDALOpenShared()** (p. **??**) it will be referenced and returned.

Starting with GDAL 1.6.0, if **GDALOpenShared()** (p. **??**) is called on the same pszFilename from two different threads, a different **GDALDataset** (p. **??**) object will be returned as it is not safe to use the same dataset from different threads, unless the user does explicitely use mutexes in its code.

See also:

GDALOpen() (p. **??**)

Parameters:

pszFilename the name of the file to access. In the case of exotic drivers this may not refer to a physical file, but instead contain information for the driver on how to access a dataset.

eAccess the desired access, either GA_Update or GA_ReadOnly. Many drivers support only read only access.

Returns:

A GDALDatasetH handle or NULL on failure. For C++ applications this handle can be cast to a **GDALDataset** (p. **??**) *.

The documentation for this class was generated from the following files:

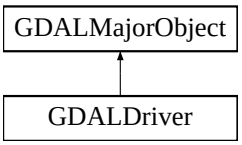
- gdal_priv.h
- gdaldataset.cpp
- rasterio.cpp

43.10 GDALDriver Class Reference

Format specific driver.

`#include <gdal_priv.h>`

Inheritance diagram for GDALDriver::



Public Member Functions

- **GDALDataset** ∗ **Create** (const char ∗pszName, int nXSize, int nYSize, int nBands, **GDALDataType** eType, char ∗∗papszOptions)
- CPLErr **Delete** (const char ∗pszName)
- CPLErr **Rename** (const char ∗pszNewName, const char ∗pszOldName)
- CPLErr **CopyFiles** (const char ∗pszNewName, const char ∗pszOldName)
- **GDALDataset** ∗ **CreateCopy** (const char ∗, **GDALDataset** ∗, int, char ∗∗, GDALProgressFunc pfnProgress, void ∗pProgressData)
- **GDALDataset** ∗ **DefaultCreateCopy** (const char ∗, **GDALDataset** ∗, int, char ∗∗, GDALProgressFunc pfnProgress, void ∗pProgressData)

Static Public Member Functions

- static CPLErr **DefaultCopyMasks** (**GDALDataset** ∗poSrcDS, **GDALDataset** ∗poDstDS, int bStrict)
- static CPLErr **QuietDelete** (const char ∗pszName)

Public Attributes

- **GDALDataset** ∗(∗ **pfnOpen**)(GDALOpenInfo ∗)
- **GDALDataset** ∗(∗ **pfnCreate**)(const char ∗pszName, int nXSize, int nYSize, int nBands, **GDALDataType** eType, char ∗∗papszOptions)
- CPLErr(∗ **pfnDelete**)(const char ∗pszName)
- **GDALDataset** ∗(∗ **pfnCreateCopy**)(const char ∗, **GDALDataset** ∗, int, char ∗∗, GDALProgressFunc pfnProgress, void ∗pProgressData)
- void ∗ **pDriverData**
- void(∗ **pfnUnloadDriver**)(**GDALDriver** ∗)
- int(∗ **pfnIdentify**)(GDALOpenInfo ∗)
- CPLErr(∗ **pfnRename**)(const char ∗pszNewName, const char ∗pszOldName)
- CPLErr(∗ **pfnCopyFiles**)(const char ∗pszNewName, const char ∗pszOldName)

43.10.1 Detailed Description

Format specific driver.

An instance of this class is created for each supported format, and manages information about the format.

This roughly corresponds to a file format, though some drivers may be gateways to many formats through a secondary multi-library.

43.10.2 Member Function Documentation

43.10.2.1 GDALDataset * GDALDriver::Create (const char * *pszFilename*, int *nXSize*, int *nYSize*, int *nBands*, GDALDataType *eType*, char ** *papszParmList*)

Create a new dataset with this driver.

What argument values are legal for particular drivers is driver specific, and there is no way to query in advance to establish legal values.

That function will try to validate the creation option list passed to the driver with the **GDALValidateCreationOptions()** (p. **??**) method. This check can be disabled by defining the configuration option GDAL_VALIDATE_CREATION_OPTIONS=NO.

Equivelent of the C function **GDALCreate()** (p. **??**).

Parameters:

pszFilename the name of the dataset to create.

nXSize width of created raster in pixels.

nYSize height of created raster in pixels.

nBands number of bands.

eType type of raster.

papszParmList list of driver specific control parameters.

Returns:

NULL on failure, or a new **GDALDataset** (p. **??**).

References GDALGetDataTypeName(), GDALValidateCreationOptions(), GDALMajorObject::GetDescription(), GDALDataset::poDriver, QuietDelete(), and GDALMajorObject::SetDescription().

Referenced by GDALCreate().

43.10.2.2 CPLErr GDALDriver::Delete (const char * *pszFilename*)

Delete named dataset.

The driver will attempt to delete the named dataset in a driver specific fashion. Full featured drivers will delete all associated files, database objects, or whatever is appropriate. The default behaviour when no driver specific behaviour is provided is to attempt to delete the passed name as a single file.

It is unwise to have open dataset handles on this dataset when it is deleted.

Equivelent of the C function **GDALDeleteDataset()** (p. **??**).

Parameters:

pszFilename name of dataset to delete.

Returns:

CE_None on success, or CE_Failure if the operation fails.

References GA_ReadOnly, GDALClose(), GDALGetFileList(), GDALOpen(), and VSIUnlink().

Referenced by QuietDelete().

43.10.2.3 CPLErr GDALDriver::Rename (const char * *pszNewName*, const char * *pszOldName*)

Rename a dataset.

Rename a dataset. This may including moving the dataset to a new directory or even a new filesystem.

It is unwise to have open dataset handles on this dataset when it is being renamed.

Equivelent of the C function **GDALRenameDataset()** (p. **??**).

Parameters:

pszNewName new name for the dataset.

pszOldName old name for the dataset.

Returns:

CE_None on success, or CE_Failure if the operation fails.

References CPLCorrespondingPaths(), GA_ReadOnly, GDALClose(), GDALGetFileList(), and GDALOpen().

43.10.2.4 CPLErr GDALDriver::CopyFiles (const char * *pszNewName*, const char * *pszOldName*)

Copy the files of a dataset.

Copy all the files associated with a dataset.

Equivelent of the C function **GDALCopyDatasetFiles()** (p. **??**).

Parameters:

pszNewName new name for the dataset.

pszOldName old name for the dataset.

Returns:

CE_None on success, or CE_Failure if the operation fails.

References CPLCorrespondingPaths(), GA_ReadOnly, GDALClose(), GDALGetFileList(), GDALOpen(), and VSIUnlink().

43.10.2.5 GDALDataset * GDALDriver::CreateCopy (const char * *pszFilename*, GDALDataset * *poSrcDS*, int *bStrict*, char ** *papszOptions*, GDALProgressFunc *pfnProgress*, void * *pProgressData*)

Create a copy of a dataset.

This method will attempt to create a copy of a raster dataset with the indicated filename, and in this drivers format. Band number, size, type, projection, geotransform and so forth are all to be copied from the provided template dataset.

Note that many sequential write once formats (such as JPEG and PNG) don't implement the **Create()** (p. **??**) method but do implement this **CreateCopy()** (p. **??**) method. If the driver doesn't implement **CreateCopy()** (p. **??**), but does implement **Create()** (p. **??**) then the default **CreateCopy()** (p. **??**) mechanism built on calling **Create()** (p. **??**) will be used.

It is intended that **CreateCopy()** (p. **??**) will often be used with a source dataset which is a virtual dataset allowing configuration of band types, and other information without actually duplicating raster data (see the VRT driver). This is what is done by the gdal_translate utility for example.

That function will try to validate the creation option list passed to the driver with the **GDALValidateCreationOptions()** (p. **??**) method. This check can be disabled by defining the configuration option GDAL_VALIDATE_CREATION_OPTIONS=NO.

Parameters:

pszFilename the name for the new dataset.

poSrcDS the dataset being duplicated.

bStrict TRUE if the copy must be strictly equivelent, or more normally FALSE indicating that the copy may adapt as needed for the output format.

papszOptions additional format dependent options controlling creation of the output file.

pfnProgress a function to be used to report progress of the copy.

pProgressData application data passed into progress function.

Returns:

a pointer to the newly created dataset (may be read-only access).

References GDALValidateCreationOptions(), GDALMajorObject::GetDescription(), GDALDataset::poDriver, QuietDelete(), and GDALMajorObject::SetDescription().

Referenced by GDALCreateCopy().

43.10.2.6 CPLErr GDALDriver::QuietDelete (const char * *pszName*) [static]

Delete dataset if found.

This is a helper method primarily used by **Create()** (p. **??**) and **CreateCopy()** (p. **??**) to predelete any dataset of the name soon to be created. It will attempt to delete the named dataset if one is found, otherwise it does nothing. An error is only returned if the dataset is found but the delete fails.

This is a static method and it doesn't matter what driver instance it is invoked on. It will attempt to discover the correct driver using Identify().

Parameters:

pszName the dataset name to try and delete.

Returns:

CE_None if the dataset does not exist, or is deleted without issues.

References Delete(), and GDALIdentifyDriver().

Referenced by Create(), and CreateCopy().

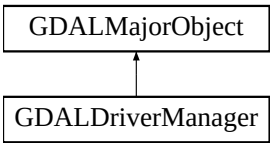
The documentation for this class was generated from the following files:

- gdal_priv.h
- gdaldriver.cpp

43.11 GDALDriverManager Class Reference

`#include <gdal_priv.h>`

Inheritance diagram for GDALDriverManager::



Public Member Functions

- int **GetDriverCount** (void)
- **GDALDriver** * **GetDriver** (int)
- **GDALDriver** * **GetDriverByName** (const char *)
- int **RegisterDriver** (**GDALDriver** *)
- void **MoveDriver** (**GDALDriver** *, int)
- void **DeregisterDriver** (**GDALDriver** *)
- void **AutoLoadDrivers** ()
- void **AutoSkipDrivers** ()
- const char * **GetHome** ()
- void **SetHome** (const char *)

43.11.1 Detailed Description

Class for managing the registration of file format drivers.

Use GetGDALDriverManager() to fetch the global singleton instance of this class.

43.11.2 Member Function Documentation

43.11.2.1 int GDALDriverManager::GetDriverCount (void)

Fetch the number of registered drivers.

This C analog to this is **GDALGetDriverCount()** (p. **??**).

Returns:

the number of registered drivers.

Referenced by GDALGetDriverCount(), GDALIdentifyDriver(), and GDALOpen().

43.11.2.2 GDALDriver * GDALDriverManager::GetDriver (int *iDriver*)

Fetch driver by index.

This C analog to this is **GDALGetDriver()** (p. **??**).

Parameters:

iDriver the driver index from 0 to **GetDriverCount()** (p. **??**)-1.

Returns:

the number of registered drivers.

Referenced by GDALGetDriver(), GDALIdentifyDriver(), and GDALOpen().

43.11.2.3 GDALDriver * GDALDriverManager::GetDriverByName (const char * *pszName*)

Fetch a driver based on the short name.

The C analog is the **GDALGetDriverByName()** (p. **??**) function.

Parameters:

pszName the short name, such as GTiff, being searched for.

Returns:

the identified driver, or NULL if no match is found.

References GDALMajorObject::GetDescription().

Referenced by AutoSkipDrivers(), GDALGetDriverByName(), and RegisterDriver().

43.11.2.4 int GDALDriverManager::RegisterDriver (GDALDriver * *poDriver*)

Register a driver for use.

The C analog is **GDALRegisterDriver()** (p. **??**).

Normally this method is used by format specific C callable registration entry points such as GDALRegister_GTiff() rather than being called directly by application level code.

If this driver (based on the object pointer, not short name) is already registered, then no change is made, and the index of the existing driver is returned. Otherwise the driver list is extended, and the new driver is added at the end.

Parameters:

poDriver the driver to register.

Returns:

the index of the new installed driver.

References GDALMajorObject::GetDescription(), GetDriverByName(), GDALDriver::pfnCreate, GDALDriver::pfnCreateCopy, and GDALMajorObject::SetMetadataItem().

Referenced by GDALRegisterDriver().

43.11.2.5 void GDALDriverManager::DeregisterDriver (GDALDriver * *poDriver*)

Deregister the passed driver.

If the driver isn't found no change is made.

The C analog is **GDALDeregisterDriver()** (p. **??**).

Parameters:

poDriver the driver to deregister.

Referenced by AutoSkipDrivers(), and GDALDeregisterDriver().

43.11.2.6 void GDALDriverManager::AutoLoadDrivers ()

Auto-load GDAL drivers from shared libraries.

This function will automatically load drivers from shared libraries. It searches the "driver path" for .so (or .dll) files that start with the prefix "gdal_X.so". It then tries to load them and then tries to call a function within them called GDALRegister_X() where the 'X' is the same as the remainder of the shared library basename ('X' is case sensitive), or failing that to call GDALRegisterMe().

There are a few rules for the driver path. If the GDAL_DRIVER_PATH environment variable it set, it is taken to be a list of directories to search separated by colons on UNIX, or semi-colons on Windows. Otherwise the /usr/local/lib/gdalplugins directory, and (if known) the lib/gdalplugins subdirectory of the gdal home directory are searched on UNIX and on Windows.

References CPLFormFilename(), CPLGetBasename(), CPLGetDirname(), CPLGetExecPath(), CPLGetExtension(), and CPLGetSymbol().

Referenced by GDALAllRegister().

43.11.2.7 void GDALDriverManager::AutoSkipDrivers ()

This method unload undesirable drivers.

All drivers specified in the space delimited list in the GDAL_SKIP environmentvariable) will be deregistered and destroyed. This method should normally be called after registration of standard drivers to allow the user a way of unloading undesired drivers. The **GDALAllRegister()** (p. **??**) function already invokes **AutoSkipDrivers()** (p. **??**) at the end, so if that functions is called, it should not be necessary to call this method from application code.

References DeregisterDriver(), and GetDriverByName().

Referenced by GDALAllRegister().

The documentation for this class was generated from the following files:

- gdal_priv.h
- gdaldrivermanager.cpp

43.12 GDALGridDataMetricsOptions Struct Reference

`#include <gdal_alg.h>`

Public Attributes

- double **dfRadius1**
- double **dfRadius2**
- double **dfAngle**
- GUInt32 **nMinPoints**
- double **dfNoDataValue**

43.12.1 Detailed Description

Data metrics method control options

43.12.2 Member Data Documentation

43.12.2.1 double GDALGridDataMetricsOptions::dfRadius1

The first radius (X axis if rotation angle is 0) of search ellipse.

43.12.2.2 double GDALGridDataMetricsOptions::dfRadius2

The second radius (Y axis if rotation angle is 0) of search ellipse.

43.12.2.3 double GDALGridDataMetricsOptions::dfAngle

Angle of ellipse rotation in degrees.

Ellipse rotated counter clockwise.

43.12.2.4 GUInt32 GDALGridDataMetricsOptions::nMinPoints

Minimum number of data points to average.

If less amount of points found the grid node considered empty and will be filled with NODATA marker.

43.12.2.5 double GDALGridDataMetricsOptions::dfNoDataValue

No data marker to fill empty points.

The documentation for this struct was generated from the following file:

- **gdal_alg.h**

43.13 GDALGridInverseDistanceToAPowerOptions Struct Reference

`#include <gdal_alg.h>`

Public Attributes

- double **dfPower**
- double **dfSmoothing**
- double **dfAnisotropyRatio**
- double **dfAnisotropyAngle**
- double **dfRadius1**
- double **dfRadius2**
- double **dfAngle**
- GUInt32 **nMaxPoints**
- GUInt32 **nMinPoints**
- double **dfNoDataValue**

43.13.1 Detailed Description

Inverse distance to a power method control options

43.13.2 Member Data Documentation

43.13.2.1 double GDALGridInverseDistanceToAPowerOptions::dfPower

Weighting power.

43.13.2.2 double GDALGridInverseDistanceToAPowerOptions::dfSmoothing

Smoothing parameter.

43.13.2.3 double GDALGridInverseDistanceToAPowerOptions::dfAnisotropyRatio

Reserved for future use.

43.13.2.4 double GDALGridInverseDistanceToAPowerOptions::dfAnisotropyAngle

Reserved for future use.

43.13.2.5 double GDALGridInverseDistanceToAPowerOptions::dfRadius1

The first radius (X axis if rotation angle is 0) of search ellipse.

43.13.2.6 double GDALGridInverseDistanceToAPowerOptions::dfRadius2

The second radius (Y axis if rotation angle is 0) of search ellipse.

43.13.2.7 double GDALGridInverseDistanceToAPowerOptions::dfAngle

Angle of ellipse rotation in degrees.

Ellipse rotated counter clockwise.

43.13.2.8 GUInt32 GDALGridInverseDistanceToAPowerOptions::nMaxPoints

Maximum number of data points to use.

Do not search for more points than this number. If less amount of points found the grid node considered empty and will be filled with NODATA marker.

43.13.2.9 GUInt32 GDALGridInverseDistanceToAPowerOptions::nMinPoints

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.

43.13.2.10 double GDALGridInverseDistanceToAPowerOptions::dfNoDataValue

No data marker to fill empty points.

The documentation for this struct was generated from the following file:

- **gdal_alg.h**

43.14 GDALGridMovingAverageOptions Struct Reference

`#include <gdal_alg.h>`

Public Attributes

- double **dfRadius1**
- double **dfRadius2**
- double **dfAngle**
- GUInt32 **nMinPoints**
- double **dfNoDataValue**

43.14.1 Detailed Description

Moving average method control options

43.14.2 Member Data Documentation

43.14.2.1 double GDALGridMovingAverageOptions::dfRadius1

The first radius (X axis if rotation angle is 0) of search ellipse.

43.14.2.2 double GDALGridMovingAverageOptions::dfRadius2

The second radius (Y axis if rotation angle is 0) of search ellipse.

43.14.2.3 double GDALGridMovingAverageOptions::dfAngle

Angle of ellipse rotation in degrees.

Ellipse rotated counter clockwise.

43.14.2.4 GUInt32 GDALGridMovingAverageOptions::nMinPoints

Minimum number of data points to average.

If less amount of points found the grid node considered empty and will be filled with NODATA marker.

43.14.2.5 double GDALGridMovingAverageOptions::dfNoDataValue

No data marker to fill empty points.

The documentation for this struct was generated from the following file:

- **gdal_alg.h**

43.15 GDALGridNearestNeighborOptions Struct Reference

`#include <gdal_alg.h>`

Public Attributes

- double **dfRadius1**
- double **dfRadius2**
- double **dfAngle**
- double **dfNoDataValue**

43.15.1 Detailed Description

Nearest neighbor method control options

43.15.2 Member Data Documentation

43.15.2.1 double GDALGridNearestNeighborOptions::dfRadius1

The first radius (X axis if rotation angle is 0) of search ellipse.

43.15.2.2 double GDALGridNearestNeighborOptions::dfRadius2

The second radius (Y axis if rotation angle is 0) of search ellipse.

43.15.2.3 double GDALGridNearestNeighborOptions::dfAngle

Angle of ellipse rotation in degrees.

Ellipse rotated counter clockwise.

43.15.2.4 double GDALGridNearestNeighborOptions::dfNoDataValue

No data marker to fill empty points.

The documentation for this struct was generated from the following file:

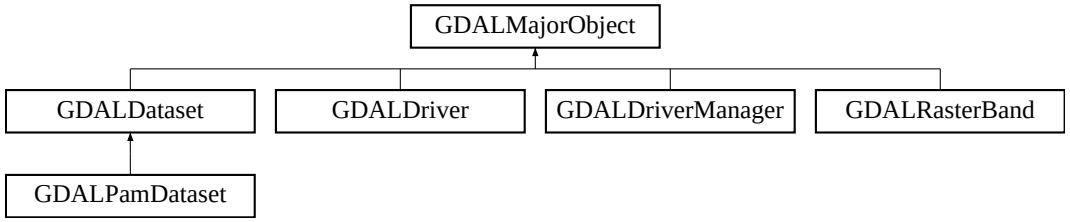
- **gdal_alg.h**

43.16 GDALMajorObject Class Reference

Object with metadata.

`#include <gdal_priv.h>`

Inheritance diagram for GDALMajorObject::



Public Member Functions

- int **GetMOFlags** ()
- void **SetMOFlags** (int nFlags)
- virtual const char ∗ **GetDescription** () const
- virtual void **SetDescription** (const char ∗)
- virtual char ∗∗ **GetMetadata** (const char ∗pszDomain="")
- virtual CPLErr **SetMetadata** (char ∗∗papszMetadata, const char ∗pszDomain="")
- virtual const char ∗ **GetMetadataItem** (const char ∗pszName, const char ∗pszDomain="")
- virtual CPLErr **SetMetadataItem** (const char ∗pszName, const char ∗pszValue, const char ∗pszDomain="")

Protected Attributes

- int **nFlags**
- CPLString **sDescription**
- GDALMultiDomainMetadata **oMDMD**

43.16.1 Detailed Description

Object with metadata.

43.16.2 Member Function Documentation

43.16.2.1 const char ∗ GDALMajorObject::GetDescription () const [virtual]

Fetch object description.

The semantics of the returned description are specific to the derived type. For GDALDatasets it is the dataset name. For GDALRasterBands it is actually a description (if supported) or "".

This method is the same as the C function **GDALGetDescription()** (p. **??**).

Returns:

pointer to internal description string.

Referenced by GDALDriver::Create(), GDALDriver::CreateCopy(), GDALOpen(), GDALOpenShared(), GDALValidateCreationOptions(), GDALDriverManager::GetDriverByName(), GDALDataset::GetFileList(), GDALRasterBand::GetLockedBlockRef(), GDALDataset::MarkAsShared(), GDALDriverManager::RegisterDriver(), GDALDataset::~GDALDataset(), and GDALRasterBand::~GDALRasterBand().

43.16.2.2 void GDALMajorObject::SetDescription (const char * *pszNewDesc*) [virtual]

Set object description.

The semantics of the description are specific to the derived type. For GDALDatasets it is the dataset name. For GDALRasterBands it is actually a description (if supported) or "".

Normally application code should not set the "description" for GDALDatasets. It is handled internally.

This method is the same as the C function **GDALSetDescription()** (p. **??**).

Referenced by GDALDriver::Create(), GDALDriver::CreateCopy(), and GDALOpen().

43.16.2.3 char ** GDALMajorObject::GetMetadata (const char * *pszDomain* = "") [virtual]

Fetch metadata.

The returned string list is owned by the object, and may change at any time. It is formated as a "Name=value" list with the last pointer value being NULL. Use the the CPL StringList functions such as CSLFetchNameValue() to manipulate it.

Note that relatively few formats return any metadata at this time.

This method does the same thing as the C function **GDALGetMetadata()** (p. **??**).

Parameters:

pszDomain the domain of interest. Use "" or NULL for the default domain.

Returns:

NULL or a string list.

43.16.2.4 CPLErr GDALMajorObject::SetMetadata (char ** *papszMetadataIn*, const char * *pszDomain* = "") [virtual]

Set metadata.

The C function **GDALSetMetadata()** (p. **??**) does the same thing as this method.

Parameters:

papszMetadata the metadata in name=value string list format to apply.

pszDomain the domain of interest. Use "" or NULL for the default domain.

Returns:

CE_None on success, CE_Failure on failure and CE_Warning if the metadata has been accepted, but is likely not maintained persistently by the underlying object between sessions.

Reimplemented in **GDALPamDataset** (p. **??**).

Referenced by GDALPamDataset::SetMetadata().

43.16.2.5 const char * GDALMajorObject::GetMetadataItem (const char * *pszName*, const char * *pszDomain* = "") [virtual]

Fetch single metadata item.

The C function **GDALGetMetadataItem()** (p. **??**) does the same thing as this method.

Parameters:

pszName the key for the metadata item to fetch.

pszDomain the domain to fetch for, use NULL for the default domain.

Returns:

NULL on failure to find the key, or a pointer to an internal copy of the value string on success.

Referenced by GDALDatasetCopyWholeRaster(), GDALRegenerateOverviews(), GDALRasterBand::GetMaximum(), GDALRasterBand::GetMinimum(), and GDALRasterBand::GetStatistics().

43.16.2.6 CPLErr GDALMajorObject::SetMetadataItem (const char * *pszName*, const char * *pszValue*, const char * *pszDomain* = "") [virtual]

Set single metadata item.

The C function **GDALSetMetadataItem()** (p. **??**) does the same thing as this method.

Parameters:

pszName the key for the metadata item to fetch.

pszValue the value to assign to the key.

pszDomain the domain to set within, use NULL for the default domain.

Returns:

CE_None on success, or an error code on failure.

Reimplemented in **GDALPamDataset** (p. **??**).

Referenced by GDALDriverManager::RegisterDriver(), GDALPamDataset::SetMetadataItem(), and GDALRasterBand::SetStatistics().

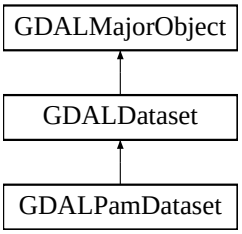
The documentation for this class was generated from the following files:

- gdal_priv.h
- gdalmajorobject.cpp

43.17 GDALPamDataset Class Reference

`#include <gdal_pam.h>`

Inheritance diagram for GDALPamDataset::



Public Member Functions

- virtual void **FlushCache** (void)
- virtual const char * **GetProjectionRef** (void)
- virtual CPLErr **SetProjection** (const char *)
- virtual CPLErr **GetGeoTransform** (double *)
- virtual CPLErr **SetGeoTransform** (double *)
- virtual int **GetGCPCount** ()
- virtual const char * **GetGCPProjection** ()
- virtual const **GDAL_GCP** * **GetGCPs** ()
- virtual CPLErr **SetGCPs** (int nGCPCount, const **GDAL_GCP** *pasGCPList, const char *pszGCPProjection)
- virtual CPLErr **SetMetadata** (char **papszMetadata, const char *pszDomain="")
- virtual CPLErr **SetMetadataItem** (const char *pszName, const char *pszValue, const char *pszDomain="")
- virtual char ** **GetFileList** (void)
- virtual CPLErr **CloneInfo** (**GDALDataset** *poSrcDS, int nCloneInfoFlags)
- void **MarkPamDirty** ()
- GDALDatasetPamInfo * **GetPamInfo** ()
- int **GetPamFlags** ()
- void **SetPamFlags** (int nValue)

Protected Member Functions

- virtual **CPLXMLNode** * **SerializeToXML** (const char *)
- virtual CPLErr **XMLInit** (**CPLXMLNode** *, const char *)
- virtual CPLErr **TryLoadXML** ()
- virtual CPLErr **TrySaveXML** ()
- CPLErr **TryLoadAux** ()
- CPLErr **TrySaveAux** ()
- virtual const char * **BuildPamFilename** ()
- void **PamInitialize** ()
- void **PamClear** ()
- void **SetPhysicalFilename** (const char *)
- void **SetSubdatasetName** (const char *)

Protected Attributes

- int **nPamFlags**
- GDALDatasetPamInfo * **psPam**

Friends

- class **GDALPamRasterBand**

43.17.1 Detailed Description

A subclass of **GDALDataset** (p. **??**) which introduces the ability to save and restore auxilary information (coordinate system, gcps, metadata, etc) not supported by a file format via an "auxilary metadata" file with the .aux.xml extension.

Enabling PAM PAM support can be enabled in GDAL by setting the GDAL_PAM_ENABLED configuration option (via CPLSetConfigOption(), or the environment) to the value of YES.

PAM Proxy Files In order to be able to record auxilary information about files on read-only media such as CDROMs or in directories where the user does not have write permissions, it is possible to enable the "PAM Proxy Database". When enabled the .aux.xml files are kept in a different directory, writable by the user.

To enable this, set the GDAL_PAM_PROXY_DIR configuration open to be the name of the directory where the proxies should be kept.

Adding PAM to Drivers Drivers for physical file formats that wish to support persistent auxilary metadata in addition to that for the format itself should derive their dataset class from **GDALPamDataset** (p. **??**) instead of directly from **GDALDataset** (p. **??**). The raster band classes should also be derived from GDALPamRasterBand.

They should also call something like this near the end of the Open() method:

```
poDS->SetDescription( poOpenInfo->pszFilename );
poDS->TryLoadXML();
```

The **SetDescription()** (p. **??**) is necessary so that the dataset will have a valid filename set as the description before TryLoadXML() is called. TryLoadXML() will look for an .aux.xml file with the same basename as the dataset and in the same directory. If found the contents will be loaded and kept track of in the **GDALPamDataset** (p. **??**) and GDALPamRasterBand objects. When a call like **GetProjectionRef()** (p. **??**) is not implemented by the format specific class, it will fall through to the PAM implementation which will return information if it was in the .aux.xml file.

Drivers should also try to call the GDALPamDataset/GDALPamRasterBand methods as a fallback if their implementation does not find information. This allows using the .aux.xml for variations that can't be stored in the format. For instance, the GeoTIFF driver **GetProjectionRef()** (p. **??**) looks like this:

```
if( EQUAL(pszProjection,"") )
    return GDALPamDataset::GetProjectionRef();
else
    return( pszProjection );
```

So if the geotiff header is missing, the .aux.xml file will be consulted.

Similarly, if **SetProjection()** (p. **??**) were called with a coordinate system not supported by GeoTIFF, the **SetProjection()** (p. **??**) method should pass it on to the **GDALPamDataset::SetProjection()** (p. **??**) method after issuing a warning that the information can't be represented within the file itself.

Drivers for subdataset based formats will also need to declare the name of the physical file they are related to, and the name of their subdataset before calling TryLoadXML().

```
poDS->SetDescription( poOpenInfo->pszFilename );
poDS->SetPhysicalFilename( poDS->pszFilename );
poDS->SetSubdatasetName( osSubdatasetName );

poDS->TryLoadXML();
```

43.17.2 Member Function Documentation

43.17.2.1 void GDALPamDataset::FlushCache (void) [virtual]

Flush all write cached data to disk.

Any raster (or other GDAL) data written via GDAL calls, but buffered internally will be written to disk.

This method is the same as the C function **GDALFlushCache()** (p. **??**).

Reimplemented from **GDALDataset** (p. **??**).

References GDALDataset::FlushCache().

43.17.2.2 const char * GDALPamDataset::GetProjectionRef (void) [virtual]

Fetch the projection definition string for this dataset.

Same as the C function **GDALGetProjectionRef()** (p. **??**).

The returned string defines the projection coordinate system of the image in OpenGIS WKT format. It should be suitable for use with the OGRSpatialReference class.

When a projection definition is not available an empty (but not NULL) string is returned.

Returns:

a pointer to an internal projection reference string. It should not be altered, freed or expected to last for long.

See also:

http://www.gdal.org/ogr/osr_tutorial.html

Reimplemented from **GDALDataset** (p. **??**).

References GDALDataset::GetProjectionRef().

43.17.2.3 CPLErr GDALPamDataset::SetProjection (const char *) [virtual]

Set the projection reference string for this dataset.

The string should be in OGC WKT or PROJ.4 format. An error may occur because of incorrectly specified projection strings, because the dataset is not writable, or because the dataset does not support the indicated projection. Many formats do not support writing projections.

This method is the same as the C **GDALSetProjection()** (p. **??**) function.

Parameters:

pszProjection projection reference string.

Returns:

CE_Failure if an error occurs, otherwise CE_None.

Reimplemented from **GDALDataset** (p. **??**).

References GDALDataset::SetProjection().

43.17.2.4 CPLErr GDALPamDataset::GetGeoTransform (double * *padfTransform*) [virtual]

Fetch the affine transformation coefficients.

Fetches the coefficients for transforming between pixel/line (P,L) raster space, and projection coordinates (Xp,Yp) space.

```
Xp = padfTransform[0] + P*padfTransform[1] + L*padfTransform[2];
Yp = padfTransform[3] + P*padfTransform[4] + L*padfTransform[5];
```

In a north up image, padfTransform[1] is the pixel width, and padfTransform[5] is the pixel height. The upper left corner of the upper left pixel is at position (padfTransform[0],padfTransform[3]).

The default transform is (0,1,0,0,0,1) and should be returned even when a CE_Failure error is returned, such as for formats that don't support transformation to projection coordinates.

NOTE: **GetGeoTransform()** (p. **??**) isn't expressive enough to handle the variety of OGC Grid Coverages pixel/line to projection transformation schemes. Eventually this method will be depreciated in favour of a more general scheme.

This method does the same thing as the C **GDALGetGeoTransform()** (p. **??**) function.

Parameters:

padfTransform an existing six double buffer into which the transformation will be placed.

Returns:

CE_None on success, or CE_Failure if no transform can be fetched.

Reimplemented from **GDALDataset** (p. **??**).

References GDALDataset::GetGeoTransform().

43.17.2.5 CPLErr GDALPamDataset::SetGeoTransform (double *) [virtual]

Set the affine transformation coefficients.

See **GetGeoTransform()** (p. **??**) for details on the meaning of the padfTransform coefficients.

This method does the same thing as the C **GDALSetGeoTransform()** (p. **??**) function.

Parameters:

padfTransform a six double buffer containing the transformation coefficients to be written with the dataset.

Returns:

CE_None on success, or CE_Failure if this transform cannot be written.

Reimplemented from **GDALDataset** (p. **??**).

References GDALDataset::SetGeoTransform().

43.17.2.6 int GDALPamDataset::GetGCPCount () [virtual]

Get number of GCPs.

This method is the same as the C function **GDALGetGCPCount()** (p. **??**).

Returns:

number of GCPs for this dataset. Zero if there are none.

Reimplemented from **GDALDataset** (p. **??**).

References GDALDataset::GetGCPCount().

43.17.2.7 const char * GDALPamDataset::GetGCPProjection () [virtual]

Get output projection for GCPs.

This method is the same as the C function **GDALGetGCPProjection()** (p. **??**).

The projection string follows the normal rules from **GetProjectionRef()** (p. **??**).

Returns:

internal projection string or "" if there are no GCPs.

Reimplemented from **GDALDataset** (p. **??**).

References GDALDataset::GetGCPProjection().

43.17.2.8 const GDAL_GCP * GDALPamDataset::GetGCPs () [virtual]

Fetch GCPs.

This method is the same as the C function **GDALGetGCPs()** (p. **??**).

Returns:

pointer to internal GCP structure list. It should not be modified, and may change on the next GDAL call.

Reimplemented from **GDALDataset** (p. **??**).

References GDALDataset::GetGCPs().

43.17.2.9 CPLErr GDALPamDataset::SetGCPs (int *nGCPCount*, const GDAL_GCP * *pasGCPList*, const char * *pszGCPProjection*) [virtual]

Assign GCPs.

This method is the same as the C function **GDALSetGCPs()** (p. **??**).

This method assigns the passed set of GCPs to this dataset, as well as setting their coordinate system. Internally copies are made of the coordinate system and list of points, so the caller remains resposible for deallocating these arguments if appropriate.

Most formats do not support setting of GCPs, even foramts that can handle GCPs. These formats will return CE_Failure.

Parameters:

nGCPCount number of GCPs being assigned.

pasGCPList array of GCP structures being assign (nGCPCount in array).

pszGCPProjection the new OGC WKT coordinate system to assign for the GCP output coordinates. This parameter should be "" if no output coordinate system is known.

Returns:

CE_None on success, CE_Failure on failure (including if action is not supported for this format).

Reimplemented from **GDALDataset** (p. **??**).

References GDALDataset::SetGCPs().

43.17.2.10 CPLErr GDALPamDataset::SetMetadata (char ** *papszMetadataIn*, const char * *pszDomain* = "") [virtual]

Set metadata.

The C function **GDALSetMetadata()** (p. **??**) does the same thing as this method.

Parameters:

papszMetadata the metadata in name=value string list format to apply.

pszDomain the domain of interest. Use "" or NULL for the default domain.

Returns:

CE_None on success, CE_Failure on failure and CE_Warning if the metadata has been accepted, but is likely not maintained persistently by the underlying object between sessions.

Reimplemented from **GDALMajorObject** (p. **??**).

References GDALMajorObject::SetMetadata().

43.17.2.11 CPLErr GDALPamDataset::SetMetadataItem (const char * *pszName*, const char * *pszValue*, const char * *pszDomain* = "") [virtual]

Set single metadata item.

The C function **GDALSetMetadataItem()** (p. **??**) does the same thing as this method.

Parameters:

pszName the key for the metadata item to fetch.

pszValue the value to assign to the key.

pszDomain the domain to set within, use NULL for the default domain.

Returns:

CE_None on success, or an error code on failure.

Reimplemented from **GDALMajorObject** (p. **??**).

References GDALMajorObject::SetMetadataItem().

43.17.2.12 char ** GDALPamDataset::GetFileList (void) [virtual]

Fetch files forming dataset.

Returns a list of files believed to be part of this dataset. If it returns an empty list of files it means there is believed to be no local file system files associated with the dataset (for instance a virtual dataset). The returned file list is owned by the caller and should be deallocated with **CSLDestroy()** (p. **??**).

The returned filenames will normally be relative or absolute paths depending on the path used to originally open the dataset.

This method is the same as the C **GDALGetFileList()** (p. **??**) function.

Returns:

NULL or a NULL terminated array of file names.

Reimplemented from **GDALDataset** (p. **??**).

References GDALDataset::GetFileList(), and VSIStatL().

The documentation for this class was generated from the following files:

- gdal_pam.h
- gdalpamdataset.cpp

43.18 GDALRasterAttributeTable Class Reference

Raster Attribute Table container.

`#include <gdal_rat.h>`

Public Member Functions

- **GDALRasterAttributeTable** ()
  Construct empty table.
- **GDALRasterAttributeTable** (const **GDALRasterAttributeTable** &)
  Copy constructor.
- **GDALRasterAttributeTable** ∗ **Clone** () const
  Copy Raster Attribute Table.
- int **GetColumnCount** () const
  Fetch table column count.
- const char ∗ **GetNameOfCol** (int) const
  Fetch name of indicated column.
- **GDALRATFieldUsage GetUsageOfCol** (int) const
  Fetch column usage value.
- **GDALRATFieldType GetTypeOfCol** (int) const
  Fetch color type.
- int **GetColOfUsage** (**GDALRATFieldUsage**) const
  Fetch column index for given usage.
- int **GetRowCount** () const
  Fetch row count.
- const char ∗ **GetValueAsString** (int iRow, int iField) const
  Fetch field value as a string.
- int **GetValueAsInt** (int iRow, int iField) const
  Fetch field value as a integer.
- double **GetValueAsDouble** (int iRow, int iField) const
  Fetch field value as a double.
- void **SetValue** (int iRow, int iField, const char ∗pszValue)
  Set field value from string.
- void **SetValue** (int iRow, int iField, double dfValue)
  Set field value from double.

- void **SetValue** (int iRow, int iField, int nValue)

  Set field value from integer.

- void **SetRowCount** (int iCount)

  Set row count.

- int **GetRowOfValue** (double dfValue) const

  Get row for pixel value.

- int **GetRowOfValue** (int nValue) const
- int **GetColorOfValue** (double dfValue, **GDALColorEntry** *psEntry) const
- double **GetRowMin** (int iRow) const
- double **GetRowMax** (int iRow) const
- CPLErr **CreateColumn** (const char *pszFieldName, **GDALRATFieldType** eFieldType, **GDALRATFieldUsage** eFieldUsage)

  Create new column.

- CPLErr **SetLinearBinning** (double dfRow0Min, double dfBinSize)

  Set linear binning information.

- int **GetLinearBinning** (double *pdfRow0Min, double *pdfBinSize) const

  Get linear binning information.

- **CPLXMLNode** * **Serialize** () const
- CPLErr **XMLInit** (**CPLXMLNode** *, const char *)
- CPLErr **InitializeFromColorTable** (const **GDALColorTable** *)

  Initialize from color table.

- **GDALColorTable** * **TranslateToColorTable** (int nEntryCount=-1)

  Translate to a color table.

- void **DumpReadable** (FILE *=NULL)

  Dump RAT in readable form.

Friends

- const char * **GDALRATGetNameOfCol** (GDALRasterAttributeTableH, int)
- const char * **GDALRATGetValueAsString** (GDALRasterAttributeTableH, int, int)

43.18.1 Detailed Description

Raster Attribute Table container.

The **GDALRasterAttributeTable** (p. **??**) (or RAT) class is used to encapsulate a table used to provide attribute information about pixel values. Each row in the table applies to a range of pixel values (or a single value in some cases), and might have attributes such as the histogram count for that range, the color pixels of that range should be drawn names of classes or any other generic information.

Raster attribute tables can be used to represent histograms, color tables, and classification information.

Each column in a raster attribute table has a name, a type (integer, floating point or string), and a GDALRATFieldUsage. The usage distinguishes columns with particular understood purposes (such as color, histogram count, name) and columns that have specific purposes not understood by the library (long label, suitability_for_growing_wheat, etc).

In the general case each row has a column indicating the minimum pixel values falling into that category, and a column indicating the maximum pixel value. These are indicated with usage values of GFU_Min, and GFU_Max. In other cases where each row is a discrete pixel value, one column of usage GFU_MinMax can be used.

In other cases all the categories are of equal size and regularly spaced and the categorization information can be determine just by knowing the value at which the categories start, and the size of a category. This is called "Linear Binning" and the information is kept specially on the raster attribute table as a whole.

RATs are normally associated with GDALRasterBands and be be queried using the **GDALRasterBand::GetDefaultRAT()** (p. **??**) method.

43.18.2 Member Function Documentation

43.18.2.1 GDALRasterAttributeTable * GDALRasterAttributeTable::Clone () const

Copy Raster Attribute Table.

Creates a new copy of an existing raster attribute table. The new copy becomes the responsibility of the caller to destroy.

This method is the same as the C function GDALRATClone().

Returns:

new copy of the RAT.

References GDALRasterAttributeTable().

43.18.2.2 int GDALRasterAttributeTable::GetColumnCount () const

Fetch table column count.

This method is the same as the C function GDALRATGetColumnCount().

Returns:

the number of columns.

Referenced by InitializeFromColorTable().

43.18.2.3 const char * GDALRasterAttributeTable::GetNameOfCol (int *iCol*) const

Fetch name of indicated column.

This method is the same as the C function GDALRATGetNameOfCol(), except that the C function returns "const char *".

Parameters:

iCol the column index (zero based).

Returns:

the column name or an empty string for invalid column numbers.

43.18.2.4 GDALRATFieldUsage GDALRasterAttributeTable::GetUsageOfCol (int *iCol*) const

Fetch column usage value.

This method is the same as the C function GDALRATGetUsageOfCol().

Parameters:

iCol the column index (zero based).

Returns:

the column usage, or GFU_Generic for improper column numbers.

References GFU_Generic.

43.18.2.5 GDALRATFieldType GDALRasterAttributeTable::GetTypeOfCol (int *iCol*) const

Fetch color type.

This method is the same as the C function GDALRATGetTypeOfCol().

Parameters:

iCol the column index (zero based).

Returns:

column type or GFT_Integer if the column index is illegal.

References GFT_Integer.

43.18.2.6 int GDALRasterAttributeTable::GetColOfUsage (GDALRATFieldUsage *eUsage*) const

Fetch column index for given usage.

Returns the index of the first column of the requested usage type, or -1 if no match is found.

This method is the same as the C function GDALRATGetUsageOfCol().

Parameters:

eUsage usage type to search for.

Returns:

column index, or -1 on failure.

Referenced by TranslateToColorTable().

43.18.2.7 int GDALRasterAttributeTable::GetRowCount () const

Fetch row count.

This method is the same as the C function GDALRATGetRowCount().

Returns:

the number of rows.

Referenced by InitializeFromColorTable().

43.18.2.8 const char * GDALRasterAttributeTable::GetValueAsString (int *iRow*, int *iField*) const

Fetch field value as a string.

The value of the requested column in the requested row is returned as a string. If the field is numeric, it is formatted as a string using default rules, so some precision may be lost.

This method is the same as the C function GDALRATGetValueAsString(), except it returns a "const char *" result.

Parameters:

iRow row to fetch (zero based).

iField column to fetch (zero based).

Returns:

field value

References GFT_Integer, GFT_Real, and GFT_String.

43.18.2.9 int GDALRasterAttributeTable::GetValueAsInt (int *iRow*, int *iField*) const

Fetch field value as a integer.

The value of the requested column in the requested row is returned as an integer. Non-integer fields will be converted to integer with the possibility of data loss.

This method is the same as the C function GDALRATGetValueAsInt().

Parameters:

iRow row to fetch (zero based).

iField column to fetch (zero based).

Returns:

field value

References GFT_Integer, GFT_Real, and GFT_String.

Referenced by TranslateToColorTable().

43.18.2.10 double GDALRasterAttributeTable::GetValueAsDouble (int *iRow*, int *iField*) const

Fetch field value as a double.

The value of the requested column in the requested row is returned as a double. Non double fields will be converted to double with the possibility of data loss.

This method is the same as the C function GDALRATGetValueAsDouble().

Parameters:

iRow row to fetch (zero based).

iField column to fetch (zero based).

Returns:

field value

References GFT_Integer, GFT_Real, and GFT_String.

43.18.2.11 void GDALRasterAttributeTable::SetValue (int *iRow*, int *iField*, const char * *pszValue*)

Set field value from string.

The indicated field (column) on the indicated row is set from the passed value. The value will be automatically converted for other field types, with a possible loss of precision.

This method is the same as the C function GDALRATSetValueAsString().

Parameters:

iRow row to fetch (zero based).

iField column to fetch (zero based).

pszValue the value to assign.

References GFT_Integer, GFT_Real, GFT_String, and SetRowCount().

Referenced by InitializeFromColorTable().

43.18.2.12 void GDALRasterAttributeTable::SetValue (int *iRow*, int *iField*, double *dfValue*)

Set field value from double.

The indicated field (column) on the indicated row is set from the passed value. The value will be automatically converted for other field types, with a possible loss of precision.

This method is the same as the C function GDALRATSetValueAsDouble().

Parameters:

iRow row to fetch (zero based).

iField column to fetch (zero based).

dfValue the value to assign.

References GFT_Integer, GFT_Real, GFT_String, and SetRowCount().

43.18.2.13 void GDALRasterAttributeTable::SetValue (int *iRow*, int *iField*, int *nValue*)

Set field value from integer.

The indicated field (column) on the indicated row is set from the passed value. The value will be automatically converted for other field types, with a possible loss of precision.

This method is the same as the C function GDALRATSetValueAsInteger().

Parameters:

iRow row to fetch (zero based).

iField column to fetch (zero based).

nValue the value to assign.

References GFT_Integer, GFT_Real, GFT_String, and SetRowCount().

43.18.2.14 void GDALRasterAttributeTable::SetRowCount (int *nNewCount*)

Set row count.

Resizes the table to include the indicated number of rows. Newly created rows will be initialized to their default values - "" for strings, and zero for numeric fields.

This method is the same as the C function GDALRATSetRowCount().

Parameters:

nNewCount the new number of rows.

References GFT_Integer, GFT_Real, and GFT_String.

Referenced by InitializeFromColorTable(), and SetValue().

43.18.2.15 int GDALRasterAttributeTable::GetRowOfValue (double *dfValue*) const

Get row for pixel value.

Given a raw pixel value, the raster attribute table is scanned to determine which row in the table applies to the pixel value. The row index is returned.

This method is the same as the C function GDALRATGetRowOfValue().

Parameters:

dfValue the pixel value.

Returns:

the row index or -1 if no row is appropriate.

References GFT_Integer, and GFT_Real.

Referenced by TranslateToColorTable().

43.18.2.16 CPLErr GDALRasterAttributeTable::CreateColumn (const char * *pszFieldName*, GDALRATFieldType *eFieldType*, GDALRATFieldUsage *eFieldUsage*)

Create new column.

If the table already has rows, all row values for the new column will be initialized to the default value ("", or zero). The new column is always created as the last column, can will be column (field) "GetColumnCount()-1" after **CreateColumn()** (p. **??**) has completed successfully.

This method is the same as the C function GDALRATCreateColumn().

Parameters:

pszFieldName the name of the field to create.

eFieldType the field type (integer, double or string).

eFieldUsage the field usage, GFU_Generic if not known.

Returns:

CE_None on success or CE_Failure if something goes wrong.

References GFT_Integer, GFT_Real, and GFT_String.

Referenced by InitializeFromColorTable().

43.18.2.17 CPLErr GDALRasterAttributeTable::SetLinearBinning (double *dfRow0MinIn*, double *dfBinSizeIn*)

Set linear binning information.

For RATs with equal sized categories (in pixel value space) that are evenly spaced, this method may be used to associate the linear binning information with the table.

This method is the same as the C function GDALRATSetLinearBinning().

Parameters:

dfRow0MinIn the lower bound (pixel value) of the first category.

dfBinSizeIn the width of each category (in pixel value units).

Returns:

CE_None on success or CE_Failure on failure.

Referenced by InitializeFromColorTable().

43.18.2.18 int GDALRasterAttributeTable::GetLinearBinning (double * *pdfRow0Min*, double * *pdfBinSize*) const

Get linear binning information.

Returns linear binning information if any is associated with the RAT.

This method is the same as the C function GDALRATGetLinearBinning().

Parameters:

pdfRow0MinIn (out) the lower bound (pixel value) of the first category.

pdfBinSizeIn (out) the width of each category (in pixel value units).

Returns:

TRUE if linear binning information exists or FALSE if there is none.

43.18.2.19 CPLErr GDALRasterAttributeTable::InitializeFromColorTable (const GDALColorTable * *poTable*)

Initialize from color table.

This method will setup a whole raster attribute table based on the contents of the passed color table. The Value (GFU_MinMax), Red (GFU_Red), Green (GFU_Green), Blue (GFU_Blue), and Alpha (GFU_-Alpha) fields are created, and a row is set for each entry in the color table.

The raster attribute table must be empty before calling **InitializeFromColorTable()** (p. **??**).

The Value fields are set based on the implicit assumption with color tables that entry 0 applies to pixel value 0, 1 to 1, etc.

This method is the same as the C function GDALRATInitializeFromColorTable().

Parameters:

poTable the color table to copy from.

CE_None on success or CE_Failure if something goes wrong.

References GDALColorEntry::c1, GDALColorEntry::c2, GDALColorEntry::c3, GDALColorEntry::c4, CreateColumn(), GDALColorTable::GetColorEntryAsRGB(), GDALColorTable::GetColorEntryCount(), GetColumnCount(), GetRowCount(), GFT_Integer, GFU_Alpha, GFU_Blue, GFU_Green, GFU_-MinMax, GFU_Red, SetLinearBinning(), SetRowCount(), and SetValue().

43.18.2.20 GDALColorTable * GDALRasterAttributeTable::TranslateToColorTable (int *nEntryCount* = -1)

Translate to a color table.

This method will attempt to create a corresponding **GDALColorTable** (p. **??**) from this raster attribute table.

This method is the same as the C function GDALRATTranslateToColorTable().

Parameters:

nEntryCount The number of entries to produce (0 to nEntryCount-1), or -1 to auto-determine the number of entries.

Returns:

the generated color table or NULL on failure.

References GDALColorEntry::c1, GDALColorEntry::c2, GDALColorEntry::c3, GDALColorEntry::c4, GetColOfUsage(), GetRowOfValue(), GetValueAsInt(), GFU_Alpha, GFU_Blue, GFU_Green, GFU_-Max, GFU_MinMax, GFU_Red, and GDALColorTable::SetColorEntry().

43.18.2.21 void GDALRasterAttributeTable::DumpReadable (FILE * *fp* = NULL)

Dump RAT in readable form.

Currently the readable form is the XML encoding ... only barely readable.

This method is the same as the C function GDALRATDumpReadable().

Parameters:

fp file to dump to or NULL for stdout.

The documentation for this class was generated from the following files:

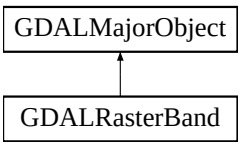
- gdal_rat.h
- gdal_rat.cpp

43.19 GDALRasterBand Class Reference

A single raster band (or channel).

`#include <gdal_priv.h>`

Inheritance diagram for GDALRasterBand::



Public Member Functions

- **GDALRasterBand** ()
- virtual ~**GDALRasterBand** ()
- int **GetXSize** ()
- int **GetYSize** ()
- int **GetBand** ()
- **GDALDataset** * **GetDataset** ()
- **GDALDataType GetRasterDataType** (void)
- void **GetBlockSize** (int *, int *)
- **GDALAccess GetAccess** ()
- CPLErr **RasterIO** (**GDALRWFlag**, int, int, int, int, void *, int, int, **GDALDataType**, int, int)
- CPLErr **ReadBlock** (int, int, void *)
- CPLErr **WriteBlock** (int, int, void *)
- GDALRasterBlock * **GetLockedBlockRef** (int nXBlockOff, int nYBlockOff, int bJustInitialize=FALSE)
- CPLErr **FlushBlock** (int=-1, int=-1)
- unsigned char * **GetIndexColorTranslationTo** (**GDALRasterBand** *poReferenceBand, unsigned char *pTranslationTable=NULL, int *pApproximateMatching=NULL)
- virtual CPLErr **FlushCache** ()
- virtual char ** **GetCategoryNames** ()
- virtual double **GetNoDataValue** (int *pbSuccess=NULL)
- virtual double **GetMinimum** (int *pbSuccess=NULL)
- virtual double **GetMaximum** (int *pbSuccess=NULL)
- virtual double **GetOffset** (int *pbSuccess=NULL)
- virtual double **GetScale** (int *pbSuccess=NULL)
- virtual const char * **GetUnitType** ()
- virtual **GDALColorInterp GetColorInterpretation** ()
- virtual **GDALColorTable** * **GetColorTable** ()
- virtual CPLErr **Fill** (double dfRealValue, double dfImaginaryValue=0)
- virtual CPLErr **SetCategoryNames** (char **)
- virtual CPLErr **SetNoDataValue** (double)
- virtual CPLErr **SetColorTable** (**GDALColorTable** *)
- virtual CPLErr **SetColorInterpretation** (**GDALColorInterp**)
- virtual CPLErr **SetOffset** (double)
- virtual CPLErr **SetScale** (double)

- virtual CPLErr **SetUnitType** (const char *)
- virtual CPLErr **GetStatistics** (int bApproxOK, int bForce, double *pdfMin, double *pdfMax, double *pdfMean, double *padfStdDev)
- virtual CPLErr **ComputeStatistics** (int bApproxOK, double *pdfMin, double *pdfMax, double *pdfMean, double *pdfStdDev, GDALProgressFunc, void *pProgressData)
- virtual CPLErr **SetStatistics** (double dfMin, double dfMax, double dfMean, double dfStdDev)
- virtual CPLErr **ComputeRasterMinMax** (int, double *)
- virtual int **HasArbitraryOverviews** ()
- virtual int **GetOverviewCount** ()
- virtual **GDALRasterBand** * **GetOverview** (int)
- virtual **GDALRasterBand** * **GetRasterSampleOverview** (int)
- virtual CPLErr **BuildOverviews** (const char *, int, int *, GDALProgressFunc, void *)
- virtual CPLErr **AdviseRead** (int nXOff, int nYOff, int nXSize, int nYSize, int nBufXSize, int nBufYSize, **GDALDataType** eDT, char **papszOptions)
- virtual CPLErr **GetHistogram** (double dfMin, double dfMax, int nBuckets, int *panHistogram, int bIncludeOutOfRange, int bApproxOK, GDALProgressFunc, void *pProgressData)
- virtual CPLErr **GetDefaultHistogram** (double *pdfMin, double *pdfMax, int *pnBuckets, int **ppanHistogram, int bForce, GDALProgressFunc, void *pProgressData)
- virtual CPLErr **SetDefaultHistogram** (double dfMin, double dfMax, int nBuckets, int *panHistogram)
- virtual const **GDALRasterAttributeTable** * **GetDefaultRAT** ()
- virtual CPLErr **SetDefaultRAT** (const **GDALRasterAttributeTable** *)
- virtual **GDALRasterBand** * **GetMaskBand** ()
- virtual int **GetMaskFlags** ()
- virtual CPLErr **CreateMaskBand** (int nFlags)

Protected Member Functions

- virtual CPLErr **IReadBlock** (int, int, void *)=0
- virtual CPLErr **IWriteBlock** (int, int, void *)
- virtual CPLErr **IRasterIO** (**GDALRWFlag**, int, int, int, int, void *, int, int, **GDALDataType**, int, int)
- CPLErr **OverviewRasterIO** (**GDALRWFlag**, int, int, int, int, void *, int, int, **GDALDataType**, int, int)
- int **InitBlockInfo** ()
- CPLErr **AdoptBlock** (int, int, GDALRasterBlock *)
- GDALRasterBlock * **TryGetLockedBlockRef** (int nXBlockOff, int nYBlockYOff)

Protected Attributes

- **GDALDataset** * **poDS**
- int **nBand**
- int **nRasterXSize**
- int **nRasterYSize**
- **GDALDataType eDataType**
- **GDALAccess eAccess**
- int **nBlockXSize**
- int **nBlockYSize**
- int **nBlocksPerRow**
- int **nBlocksPerColumn**

- int **bSubBlockingActive**
- int **nSubBlocksPerRow**
- int **nSubBlocksPerColumn**
- GDALRasterBlock ∗∗ **papoBlocks**
- int **nBlockReads**
- int **bForceCachedIO**
- **GDALRasterBand** ∗ **poMask**
- bool **bOwnMask**
- int **nMaskFlags**

Friends

- class **GDALDataset**
- class **GDALRasterBlock**
- class **GDALProxyRasterBand**

43.19.1 Detailed Description

A single raster band (or channel).

43.19.2 Constructor & Destructor Documentation

43.19.2.1 GDALRasterBand::GDALRasterBand ()

Constructor. Applications should never create GDALRasterBands directly.

References GA_ReadOnly, and GDT_Byte.

43.19.2.2 GDALRasterBand::∼GDALRasterBand () [virtual]

Destructor. Applications should never destroy GDALRasterBands directly, instead destroy the **GDALDataset** (p. **??**).

References FlushCache(), and GDALMajorObject::GetDescription().

43.19.3 Member Function Documentation

43.19.3.1 GDALRasterBlock ∗ GDALRasterBand::TryGetLockedBlockRef (int *nXBlockOff*, int *nYBlockOff*) [protected]

Try fetching block ref.

This method will returned the requested block (locked) if it is already in the block cache for the layer. If not, NULL is returned.

If a non-NULL value is returned, then a lock for the block will have been acquired on behalf of the caller. It is absolutely imperative that the caller release this lock (with GDALRasterBlock::DropLock()) or else severe problems may result.

Parameters:

nBlockXOff the horizontal block offset, with zero indicating the left most block, 1 the next block and so forth.

nYBlockOff the vertical block offset, with zero indicating the top most block, 1 the next block and so forth.

Returns:

NULL if block not available, or locked block pointer.

Referenced by GetLockedBlockRef().

43.19.3.2 int GDALRasterBand::GetXSize ()

Fetch XSize of raster.

This method is the same as the C function **GDALGetRasterBandXSize()** (p. **??**).

Returns:

the width in pixels of this band.

Referenced by ComputeStatistics(), GDALRegenerateOverviews(), GetHistogram(), GetRasterSampleOverview(), and WriteBlock().

43.19.3.3 int GDALRasterBand::GetYSize ()

Fetch YSize of raster.

This method is the same as the C function **GDALGetRasterBandYSize()** (p. **??**).

Returns:

the height in pixels of this band.

Referenced by ComputeStatistics(), GDALRegenerateOverviews(), GetHistogram(), GetRasterSampleOverview(), and WriteBlock().

43.19.3.4 int GDALRasterBand::GetBand ()

Fetch the band number.

This method returns the band that this **GDALRasterBand** (p. **??**) objects represents within it's dataset. This method may return a value of 0 to indicate **GDALRasterBand** (p. **??**) objects without an apparently relationship to a dataset, such as GDALRasterBands serving as overviews.

This method is the same as the C function **GDALGetBandNumber()** (p. **??**).

Returns:

band number (1+) or 0 if the band number isn't known.

Referenced by GDALRegenerateOverviews().

43.19.3.5 GDALDataset * GDALRasterBand::GetDataset ()

Fetch the owning dataset handle.

Note that some GDALRasterBands are not considered to be a part of a dataset, such as overviews or other "freestanding" bands.

There is currently no C analog to this method.

Returns:

the pointer to the **GDALDataset** (p. **??**) to which this band belongs, or NULL if this cannot be determined.

Referenced by GDALRegenerateOverviews().

43.19.3.6 GDALDataType GDALRasterBand::GetRasterDataType (void)

Fetch the pixel data type for this band.

Returns:

the data type of pixels for this band.

Referenced by GDALCreateWarpedVRT(), GDALDatasetCopyWholeRaster(), GDALGetRasterDataType(), GDALRasterizeGeometries(), GDALRegenerateOverviews(), GetDefaultHistogram(), and GetIndexColorTranslationTo().

43.19.3.7 void GDALRasterBand::GetBlockSize (int * *pnXSize*, int * *pnYSize*)

Fetch the "natural" block size of this band.

GDAL contains a concept of the natural block size of rasters so that applications can organized data access efficiently for some file formats. The natural block size is the block size that is most efficient for accessing the format. For many formats this is simple a whole scanline in which case *pnXSize is set to **GetXSize()** (p. **??**), and *pnYSize is set to 1.

However, for tiled images this will typically be the tile size.

Note that the X and Y block sizes don't have to divide the image size evenly, meaning that right and bottom edge blocks may be incomplete. See **ReadBlock()** (p. **??**) for an example of code dealing with these issues.

Parameters:

pnXSize integer to put the X block size into or NULL.

pnYSize integer to put the Y block size into or NULL.

Referenced by GDALDatasetCopyWholeRaster(), GDALGetBlockSize(), and GDALRegenerateOverviews().

43.19.3.8 GDALAccess GDALRasterBand::GetAccess ()

Find out if we have update permission for this band.

This method is the same as the C function **GDALGetRasterAccess()** (p. **??**).

Returns:

Either GA_Update or GA_ReadOnly.

43.19.3.9 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*)

Read/write a region of image data for this band.

This method allows reading a region of a **GDALRasterBand** (p. **??**) into a buffer, or writing data from a buffer into a region of a **GDALRasterBand** (p. **??**). It automatically takes care of data type translation if the data type (eBufType) of the buffer is different than that of the **GDALRasterBand** (p. **??**). The method also takes care of image decimation / replication if the buffer size (nBufXSize x nBufYSize) is different than the size of the region being accessed (nXSize x nYSize).

The nPixelSpace and nLineSpace parameters allow reading into or writing from unusually organized buffers. This is primarily used for buffers containing more than one bands raster data in interleaved format.

Some formats may efficiently implement decimation into a buffer by reading from lower resolution overview images.

For highest performance full resolution data access, read and write on "block boundaries" as returned by **GetBlockSize()** (p. **??**), or use the **ReadBlock()** (p. **??**) and **WriteBlock()** (p. **??**) methods.

This method is the same as the C **GDALRasterIO()** (p. **??**) function.

Parameters:

eRWFlag Either GF_Read to read a region of data, or GF_Write to write a region of data.

nXOff The pixel offset to the top left corner of the region of the band to be accessed. This would be zero to start from the left side.

nYOff The line offset to the top left corner of the region of the band to be accessed. This would be zero to start from the top.

nXSize The width of the region of the band to be accessed in pixels.

nYSize The height of the region of the band to be accessed in lines.

pData The buffer into which the data should be read, or from which it should be written. This buffer must contain at least nBufXSize * nBufYSize words of type eBufType. It is organized in left to right, top to bottom pixel order. Spacing is controlled by the nPixelSpace, and nLineSpace parameters.

nBufXSize the width of the buffer image into which the desired region is to be read, or from which it is to be written.

nBufYSize the height of the buffer image into which the desired region is to be read, or from which it is to be written.

eBufType the type of the pixel values in the pData data buffer. The pixel values will automatically be translated to/from the **GDALRasterBand** (p. **??**) data type as needed.

nPixelSpace The byte offset from the start of one pixel value in pData to the start of the next pixel value within a scanline. If defaulted (0) the size of the datatype eBufType is used.

nLineSpace The byte offset from the start of one scanline in pData to the start of the next. If defaulted the size of the datatype eBufType * nBufXSize is used.

Returns:

CE_Failure if the access fails, otherwise CE_None.

References GDALGetDataTypeSize(), GF_Read, and GF_Write.

Referenced by GDALRasterIO(), and GDALRegenerateOverviews().

43.19.3.10 CPLErr GDALRasterBand::ReadBlock (int *nXBlockOff*, int *nYBlockOff*, void * *pImage*)

Read a block of image data efficiently.

This method accesses a "natural" block from the raster band without resampling, or data type conversion. For a more generalized, but potentially less efficient access use **RasterIO()** (p. **??**).

This method is the same as the C **GDALReadBlock()** (p. **??**) function.

See the **GetLockedBlockRef()** (p. **??**) method for a way of accessing internally cached block oriented data without an extra copy into an application buffer.

Parameters:

nXBlockOff the horizontal block offset, with zero indicating the left most block, 1 the next block and so forth.

nYBlockOff the vertical block offset, with zero indicating the left most block, 1 the next block and so forth.

pImage the buffer into which the data will be read. The buffer must be large enough to hold GetBlockXSize()*GetBlockYSize() words of type **GetRasterDataType()** (p. **??**).

Returns:

CE_None on success or CE_Failure on an error.

The following code would efficiently compute a histogram of eight bit raster data. Note that the final block may be partial ... data beyond the edge of the underlying raster band in these edge blocks is of an undermined value.

```
CPLErr GetHistogram( GDALRasterBand *poBand, int *panHistogram )

{
    int        nXBlocks, nYBlocks, nXBlockSize, nYBlockSize;
    int        iXBlock, iYBlock;
    GByte      *pabyData;

    memset( panHistogram, 0, sizeof(int) * 256 );

    CPLAssert( poBand->GetRasterDataType() (p.??) == GDT_Byte );

    poBand->GetBlockSize( &nXBlockSize, &nYBlockSize );
    nXBlocks = (poBand->GetXSize() (p.??) + nXBlockSize - 1) / nXBlockSize;
    nYBlocks = (poBand->GetYSize() (p.??) + nYBlockSize - 1) / nYBlockSize;

    pabyData = (GByte *) CPLMalloc(nXBlockSize * nYBlockSize);

    for( iYBlock = 0; iYBlock < nYBlocks; iYBlock++ )
    {
        for( iXBlock = 0; iXBlock < nXBlocks; iXBlock++ )
        {
            int        nXValid, nYValid;
```

```
            poBand->ReadBlock( iXBlock, iYBlock, pabyData );

            // Compute the portion of the block that is valid
            // for partial edge blocks.
            if( (iXBlock+1) * nXBlockSize > poBand->GetXSize() (p. ??) )
                nXValid = poBand->GetXSize() (p. ??) - iXBlock * nXBlockSize;
            else
                nXValid = nXBlockSize;

            if( (iYBlock+1) * nYBlockSize > poBand->GetYSize() (p. ??) )
                nYValid = poBand->GetYSize() (p. ??) - iYBlock * nYBlockSize;
            else
                nYValid = nYBlockSize;

            // Collect the histogram counts.
            for( int iY = 0; iY < nYValid; iY++ )
            {
                for( int iX = 0; iX < nXValid; iX++ )
                {
                    panHistogram[pabyData[iX + iY * nXBlockSize]] += 1;
                }
            }
        }
    }
}
```

Referenced by GDALReadBlock().

43.19.3.11 CPLErr GDALRasterBand::WriteBlock (int *nXBlockOff*, int *nYBlockOff*, void * *pImage*)

Write a block of image data efficiently.

This method accesses a "natural" block from the raster band without resampling, or data type conversion. For a more generalized, but potentially less efficient access use **RasterIO()** (p. **??**).

This method is the same as the C **GDALWriteBlock()** (p. **??**) function.

See **ReadBlock()** (p. **??**) for an example of block oriented data access.

Parameters:

nXBlockOff the horizontal block offset, with zero indicating the left most block, 1 the next block and so forth.

nYBlockOff the vertical block offset, with zero indicating the left most block, 1 the next block and so forth.

pImage the buffer from which the data will be written. The buffer must be large enough to hold GetBlockXSize()*GetBlockYSize() words of type **GetRasterDataType()** (p. **??**).

Returns:

CE_None on success or CE_Failure on an error.

The following code would efficiently compute a histogram of eight bit raster data. Note that the final block may be partial ... data beyond the edge of the underlying raster band in these edge blocks is of an undermined value.

References GA_ReadOnly, GetXSize(), and GetYSize().

Referenced by GDALWriteBlock().

43.19.3.12 GDALRasterBlock * GDALRasterBand::GetLockedBlockRef (int *nXBlockOff*, int *nYBlockOff*, int *bJustInitialize* = FALSE)

Fetch a pointer to an internally cached raster block.

This method will returned the requested block (locked) if it is already in the block cache for the layer. If not, the block will be read from the driver, and placed in the layer block cached, then returned. If an error occurs reading the block from the driver, a NULL value will be returned.

If a non-NULL value is returned, then a lock for the block will have been acquired on behalf of the caller. It is absolutely imperative that the caller release this lock (with GDALRasterBlock::DropLock()) or else severe problems may result.

Note that calling **GetLockedBlockRef()** (p. **??**) on a previously uncached band will enable caching.

Parameters:

nBlockXOff the horizontal block offset, with zero indicating the left most block, 1 the next block and so forth.

nYBlockOff the vertical block offset, with zero indicating the top most block, 1 the next block and so forth.

bJustInitialize If TRUE the block will be allocated and initialized, but not actually read from the source. This is useful when it will just be completely set and written back.

Returns:

pointer to the block object, or NULL on failure.

References GDALMajorObject::GetDescription(), and TryGetLockedBlockRef().

Referenced by ComputeStatistics(), Fill(), and GetHistogram().

43.19.3.13 unsigned char * GDALRasterBand::GetIndexColorTranslationTo (GDALRasterBand * *poReferenceBand*, unsigned char * *pTranslationTable* = NULL, int * *pApproximateMatching* = NULL)

When the raster band has a palette index, it may be usefull to compute the "translation" of this palette to the palette of another band. The translation tries to do exact matching first, and then approximate matching if no exact matching is possible. This method returns a table such that table[i] = j where i is an index of the 'this' rasterband and j the corresponding index for the reference rasterband.

This method is thought as internal to GDAL and is used for drivers like RPFTOC.

The implementation only supports 1-byte palette rasterbands.

Parameters:

poReferenceBand the raster band

pTranslationTable an already allocated translation table (at least 256 bytes), or NULL to let the method allocate it

poApproximateMatching a pointer to a flag that is set if the matching is approximate. May be NULL.

Returns:

a translation table if the two bands are palette index and that they do not match or NULL in other cases. The table must be freed with CPLFree if NULL was passed for pTranslationTable.

References GDALColorEntry::c1, GDALColorEntry::c2, GDALColorEntry::c3, GCI_PaletteIndex, GDT_Byte, GDALColorTable::GetColorEntry(), GDALColorTable::GetColorEntryCount(), GetColorInterpretation(), GetColorTable(), GetNoDataValue(), and GetRasterDataType().

43.19.3.14 CPLErr GDALRasterBand::FlushCache (void) [virtual]

Flush raster data cache.

This call will recover memory used to cache data blocks for this raster band, and ensure that new requests are referred to the underlying driver.

This method is the same as the C function **GDALFlushRasterCache()** (p. **??**).

Returns:

CE_None on success.

Referenced by GDALDataset::FlushCache(), GDALRegenerateOverviews(), and ~GDALRasterBand().

43.19.3.15 char ** GDALRasterBand::GetCategoryNames () [virtual]

Fetch the list of category names for this raster.

The return list is a "StringList" in the sense of the CPL functions. That is a NULL terminated array of strings. Raster values without associated names will have an empty string in the returned list. The first entry in the list is for raster values of zero, and so on.

The returned stringlist should not be altered or freed by the application. It may change on the next GDAL call, so please copy it if it is needed for any period of time.

Returns:

list of names, or NULL if none.

43.19.3.16 double GDALRasterBand::GetNoDataValue (int * *pbSuccess* = NULL) [virtual]

Fetch the no data value for this band.

If there is no out of data value, an out of range value will generally be returned. The no data value for a band is generally a special marker value used to mark pixels that are not valid data. Such pixels should generally not be displayed, nor contribute to analysis operations.

This method is the same as the C function **GDALGetRasterNoDataValue()** (p. **??**).

Parameters:

pbSuccess pointer to a boolean to use to indicate if a value is actually associated with this layer. May be NULL (default).

Returns:

the nodata value for this band.

Referenced by ComputeStatistics(), GDALRegenerateOverviews(), and GetIndexColorTranslationTo().

43.19.3.17 double GDALRasterBand::GetMinimum (int * *pbSuccess* = NULL) [virtual]

Fetch the minimum value for this band.

For file formats that don't know this intrinsically, the minimum supported value for the data type will generally be returned.

This method is the same as the C function **GDALGetRasterMinimum()** (p. **??**).

Parameters:

pbSuccess pointer to a boolean to use to indicate if the returned value is a tight minimum or not. May be NULL (default).

Returns:

the minimum raster value (excluding no data pixels)

References CPLAtofM(), GDT_Byte, GDT_Float32, GDT_Float64, GDT_Int16, GDT_Int32, GDT_UInt16, GDT_UInt32, and GDALMajorObject::GetMetadataItem().

Referenced by GetStatistics().

43.19.3.18 double GDALRasterBand::GetMaximum (int * *pbSuccess* = NULL) [virtual]

Fetch the maximum value for this band.

For file formats that don't know this intrinsically, the maximum supported value for the data type will generally be returned.

This method is the same as the C function **GDALGetRasterMaximum()** (p. **??**).

Parameters:

pbSuccess pointer to a boolean to use to indicate if the returned value is a tight maximum or not. May be NULL (default).

Returns:

the maximum raster value (excluding no data pixels)

References CPLAtofM(), GDT_Byte, GDT_CFloat32, GDT_CFloat64, GDT_CInt16, GDT_CInt32, GDT_Float32, GDT_Float64, GDT_Int16, GDT_Int32, GDT_UInt16, GDT_UInt32, and GDALMajorObject::GetMetadataItem().

Referenced by GetStatistics().

43.19.3.19 double GDALRasterBand::GetOffset (int * *pbSuccess* = NULL) [virtual]

Fetch the raster value offset.

This value (in combination with the **GetScale()** (p. **??**) value) is used to transform raw pixel values into the units returned by GetUnits(). For example this might be used to store elevations in GUInt16 bands with a precision of 0.1, and starting from -100.

Units value = (raw value * scale) + offset

For file formats that don't know this intrinsically a value of zero is returned.

This method is the same as the C function GDALGetRasterOffset().

Parameters:

pbSuccess pointer to a boolean to use to indicate if the returned value is meaningful or not. May be NULL (default).

Returns:

the raster offset.

43.19.3.20 double GDALRasterBand::GetScale (int * *pbSuccess* = NULL) [virtual]

Fetch the raster value scale.

This value (in combination with the **GetOffset()** (p. **??**) value) is used to transform raw pixel values into the units returned by GetUnits(). For example this might be used to store elevations in GUInt16 bands with a precision of 0.1, and starting from -100.

Units value = (raw value * scale) + offset

For file formats that don't know this intrinsically a value of one is returned.

This method is the same as the C function GDALGetRasterScale().

Parameters:

pbSuccess pointer to a boolean to use to indicate if the returned value is meaningful or not. May be NULL (default).

Returns:

the raster scale.

43.19.3.21 const char * GDALRasterBand::GetUnitType () [virtual]

Return raster unit type.

Return a name for the units of this raster's values. For instance, it might be "m" for an elevation model in meters, or "ft" for feet. If no units are available, a value of "" will be returned. The returned string should not be modified, nor freed by the calling application.

This method is the same as the C function **GDALGetRasterUnitType()** (p. **??**).

Returns:

unit name string.

43.19.3.22 GDALColorInterp GDALRasterBand::GetColorInterpretation () [virtual]

How should this band be interpreted as color?

CV_Undefined is returned when the format doesn't know anything about the color interpretation.

This method is the same as the C function **GDALGetRasterColorInterpretation()** (p. **??**).

Returns:

color interpretation value for band.

Referenced by GDALRegenerateOverviews(), and GetIndexColorTranslationTo().

43.19.3.23 GDALColorTable * GDALRasterBand::GetColorTable () [virtual]

Fetch the color table associated with band.

If there is no associated color table, the return result is NULL. The returned color table remains owned by the **GDALRasterBand** (p. **??**), and can't be depended on for long, nor should it ever be modified by the caller.

This method is the same as the C function **GDALGetRasterColorTable()** (p. **??**).

Returns:

internal color table, or NULL.

Referenced by GDALRegenerateOverviews(), and GetIndexColorTranslationTo().

43.19.3.24 CPLErr GDALRasterBand::Fill (double *dfRealValue*, double *dfImaginaryValue* = 0) [virtual]

Fill this band with a constant value. GDAL makes no guarantees about what values pixels in newly created files are set to, so this method can be used to clear a band to a specified "default" value. The fill value is passed in as a double but this will be converted to the underlying type before writing to the file. An optional second argument allows the imaginary component of a complex constant value to be specified.

Parameters:

dfRealvalue Real component of fill value

dfImaginaryValue Imaginary component of fill value, defaults to zero

Returns:

CE_Failure if the write fails, otherwise CE_None

References GA_ReadOnly, GDALCopyWords(), GDALGetDataTypeSize(), GDT_CFloat64, and GetLockedBlockRef().

43.19.3.25 CPLErr GDALRasterBand::SetCategoryNames (char **) [virtual]

Set the category names for this band.

See the **GetCategoryNames()** (p. **??**) method for more on the interpretation of category names.

This method is the same as the C function **GDALSetRasterCategoryNames()** (p. **??**).

Parameters:

papszNames the NULL terminated StringList of category names. May be NULL to just clear the existing list.

Returns:

CE_None on success of CE_Failure on failure. If unsupported by the driver CE_Failure is returned, but no error message is reported.

43.19.3.26 CPLErr GDALRasterBand::SetNoDataValue (double) [virtual]

Set the no data value for this band.

To clear the nodata value, just set it with an "out of range" value. Complex band no data values must have an imagery component of zero.

This method is the same as the C function **GDALSetRasterNoDataValue()** (p. **??**).

Parameters:

dfNoData the value to set.

Returns:

CE_None on success, or CE_Failure on failure. If unsupported by the driver, CE_Failure is returned by no error message will have been emitted.

43.19.3.27 CPLErr GDALRasterBand::SetColorTable (GDALColorTable * *poCT*) [virtual]

Set the raster color table.

The driver will make a copy of all desired data in the colortable. It remains owned by the caller after the call.

This method is the same as the C function **GDALSetRasterColorTable()** (p. **??**).

Parameters:

poCT the color table to apply. This may be NULL to clear the color table (where supported).

Returns:

CE_None on success, or CE_Failure on failure. If the action is unsupported by the driver, a value of CE_Failure is returned, but no error is issued.

43.19.3.28 CPLErr GDALRasterBand::SetColorInterpretation (GDALColorInterp *eColorInterp*) [virtual]

Set color interpretation of a band.

Parameters:

eColorInterp the new color interpretation to apply to this band.

Returns:

CE_None on success or CE_Failure if method is unsupported by format.

43.19.3.29 CPLErr GDALRasterBand::SetOffset (double *dfNewOffset*) [virtual]

Set scaling offset.

Very few formats implement this method. When not implemented it will issue a CPLE_NotSupported error and return CE_Failure.

Parameters:

dfNewOffset the new offset.

Returns:

CE_None or success or CE_Failure on failure.

43.19.3.30 CPLErr GDALRasterBand::SetScale (double *dfNewScale*) [virtual]

Set scaling ratio.

Very few formats implement this method. When not implemented it will issue a CPLE_NotSupported error and return CE_Failure.

Parameters:

dfNewScale the new scale.

Returns:

CE_None or success or CE_Failure on failure.

43.19.3.31 CPLErr GDALRasterBand::SetUnitType (const char * *pszNewValue*) [virtual]

Set unit type.

Set the unit type for a raster band. Values should be one of "" (the default indicating it is unknown), "m" indicating meters, or "ft" indicating feet, though other nonstandard values are allowed.

Parameters:

pszNewValue the new unit type value.

Returns:

CE_None on success or CE_Failure if not succuessful, or unsupported.

43.19.3.32 CPLErr GDALRasterBand::GetStatistics (int *bApproxOK*, int *bForce*, double * *pdfMin*, double * *pdfMax*, double * *pdfMean*, double * *pdfStdDev*) [virtual]

Fetch image statistics.

Returns the minimum, maximum, mean and standard deviation of all pixel values in this band. If approximate statistics are sufficient, the bApproxOK flag can be set to true in which case overviews, or a subset of image tiles may be used in computing the statistics.

If bForce is FALSE results will only be returned if it can be done quickly (ie. without scanning the data). If bForce is FALSE and results cannot be returned efficiently, the method will return CE_Warning but no warning will have been issued. This is a non-standard use of the CE_Warning return value to indicate "nothing done".

Note that file formats using PAM (Persistent Auxilary Metadata) services will generally cache statistics in the .pam file allowing fast fetch after the first request.

This method is the same as the C function GDALGetRasterStatistics().

Parameters:

bApproxOK If TRUE statistics may be computed based on overviews or a subset of all tiles.

bForce If FALSE statistics will only be returned if it can be done without rescanning the image.

pdfMin Location into which to load image minimum (may be NULL).

pdfMax Location into which to load image maximum (may be NULL).-

pdfMean Location into which to load image mean (may be NULL).

pdfStdDev Location into which to load image standard deviation (may be NULL).

Returns:

CE_None on success, CE_Warning if no values returned, CE_Failure if an error occurs.

References ComputeStatistics(), GetMaximum(), GDALMajorObject::GetMetadataItem(), and GetMinimum().

Referenced by GetDefaultHistogram().

43.19.3.33 CPLErr GDALRasterBand::ComputeStatistics (int *bApproxOK*, double * *pdfMin*, double * *pdfMax*, double * *pdfMean*, double * *pdfStdDev*, GDALProgressFunc *pfnProgress*, void * *pProgressData*) [virtual]

Compute image statistics.

Returns the minimum, maximum, mean and standard deviation of all pixel values in this band. If approximate statistics are sufficient, the bApproxOK flag can be set to true in which case overviews, or a subset of image tiles may be used in computing the statistics.

Once computed, the statistics will generally be "set" back on the raster band using **SetStatistics()** (p. **??**).

This method is the same as the C function GDALComputeRasterStatistics().

Parameters:

bApproxOK If TRUE statistics may be computed based on overviews or a subset of all tiles.

pdfMin Location into which to load image minimum (may be NULL).

pdfMax Location into which to load image maximum (may be NULL).-

pdfMean Location into which to load image mean (may be NULL).

pdfStdDev Location into which to load image standard deviation (may be NULL).

pfnProgress a function to call to report progress, or NULL.

pProgressData application data to pass to the progress function.

Returns:

CE_None on success, or CE_Failure if an error occurs or processing is terminated by the user.

References ComputeStatistics(), GDALGetDataTypeSize(), GDT_Byte, GDT_CFloat32, GDT_-CFloat64, GDT_CInt16, GDT_CInt32, GDT_Float32, GDT_Float64, GDT_Int16, GDT_Int32, GDT_-UInt16, GDT_UInt32, GetLockedBlockRef(), GetNoDataValue(), GetOverviewCount(), GetRasterSampleOverview(), GetXSize(), GetYSize(), GF_Read, HasArbitraryOverviews(), and SetStatistics().

Referenced by ComputeStatistics(), and GetStatistics().

43.19.3.34 CPLErr GDALRasterBand::SetStatistics (double *dfMin*, double *dfMax*, double *dfMean*, double *dfStdDev*) [virtual]

Set statistics on band.

This method can be used to store min/max/mean/standard deviation statistics on a raster band.

The default implementation stores them as metadata, and will only work on formats that can save arbitrary metadata. This method cannot detect whether metadata will be properly saved and so may return CE_None even if the statistics will never be saved.

This method is the same as the C function GDALSetRasterStatistics().

Parameters:

dfMin minimum pixel value.

dfMax maximum pixel value.

dfMean mean (average) of all pixel values.

dfStdDev Standard deviation of all pixel values.

Returns:

CE_None on success or CE_Failure on failure.

References GDALMajorObject::SetMetadataItem().

Referenced by ComputeStatistics().

43.19.3.35 int GDALRasterBand::HasArbitraryOverviews () [virtual]

Check for arbitrary overviews.

This returns TRUE if the underlying datastore can compute arbitrary overviews efficiently, such as is the case with OGDI over a network. Datastores with arbitrary overviews don't generally have any fixed overviews, but the **RasterIO()** (p. **??**) method can be used in downsampling mode to get overview data efficiently.

This method is the same as the C function **GDALHasArbitraryOverviews()** (p. **??**),

Returns:

TRUE if arbitrary overviews available (efficiently), otherwise FALSE.

Referenced by ComputeStatistics(), and GetHistogram().

43.19.3.36 int GDALRasterBand::GetOverviewCount () [virtual]

Return the number of overview layers available.

This method is the same as the C function **GDALGetOverviewCount()** (p. **??**).

Returns:

overview count, zero if none.

References GDALDataset::oOvManager.

Referenced by ComputeStatistics(), GetHistogram(), and GetRasterSampleOverview().

43.19.3.37 GDALRasterBand * GDALRasterBand::GetOverview (int *i*) [virtual]

Fetch overview raster band object.

This method is the same as the C function **GDALGetOverview()** (p. **??**).

Parameters:

i overview index between 0 and **GetOverviewCount()** (p. **??**)-1.

Returns:

overview **GDALRasterBand** (p. **??**).

References GDALDataset::oOvManager.

Referenced by GetRasterSampleOverview().

43.19.3.38 GDALRasterBand * GDALRasterBand::GetRasterSampleOverview (int *nDesiredSamples*) [virtual]

Fetch best sampling overview.

Returns the most reduced overview of the given band that still satisfies the desired number of samples. This function can be used with zero as the number of desired samples to fetch the most reduced overview. The same band as was passed in will be returned if it has not overviews, or if none of the overviews have enough samples.

This method is the same as the C function GDALGetRasterSampleOverview().

Parameters:

nDesiredSamples the returned band will have at least this many pixels.

Returns:

optimal overview or the band itself.

References GetOverview(), GetOverviewCount(), GetXSize(), and GetYSize().

Referenced by ComputeStatistics(), and GetHistogram().

43.19.3.39 CPLErr GDALRasterBand::BuildOverviews (const char * *pszResampling*, int *nOverviews*, int * *panOverviewList*, GDALProgressFunc *pfnProgress*, void * *pProgressData*) [virtual]

Build raster overview(s)

If the operation is unsupported for the indicated dataset, then CE_Failure is returned, and **CPLGetLastErrorNo()** (p. **??**) will return CPLE_NotSupported.

WARNING: It is not possible to build overviews for a single band in TIFF format, and thus this method does not work for TIFF format, or any formats that use the default overview building in TIFF format. Instead it is necessary to build overviews on the dataset as a whole using **GDALDataset::BuildOverviews()** (p. **??**). That makes this method pretty useless from a practical point of view.

Parameters:

pszResampling one of "NEAREST", "AVERAGE", or "AVERAGE_MAGPHASE", "GAUSS" or "NONE" controlling the downsampling method applied.

nOverviews number of overviews to build.

panOverviewList the list of overview decimation factors to build.

pfnProgress a function to call to report progress, or NULL.

pProgressData application data to pass to the progress function.

Returns:

CE_None on success or CE_Failure if the operation doesn't work.

43.19.3.40 CPLErr GDALRasterBand::AdviseRead (int *nXOff*, int *nYOff*, int *nXSize*, int *nYSize*, int *nBufXSize*, int *nBufYSize*, GDALDataType *eDT*, char ∗∗ *papszOptions*) [virtual]

Advise driver of upcoming read requests.

Some GDAL drivers operate more efficiently if they know in advance what set of upcoming read requests will be made. The **AdviseRead()** (p. **??**) method allows an application to notify the driver of the region of interest, and at what resolution the region will be read.

Many drivers just ignore the **AdviseRead()** (p. **??**) call, but it can dramatically accelerate access via some drivers.

Parameters:

nXOff The pixel offset to the top left corner of the region of the band to be accessed. This would be zero to start from the left side.

nYOff The line offset to the top left corner of the region of the band to be accessed. This would be zero to start from the top.

nXSize The width of the region of the band to be accessed in pixels.

nYSize The height of the region of the band to be accessed in lines.

nBufXSize the width of the buffer image into which the desired region is to be read, or from which it is to be written.

nBufYSize the height of the buffer image into which the desired region is to be read, or from which it is to be written.

eBufType the type of the pixel values in the pData data buffer. The pixel values will automatically be translated to/from the **GDALRasterBand** (p. **??**) data type as needed.

papszOptions a list of name=value strings with special control options. Normally this is NULL.

Returns:

CE_Failure if the request is invalid and CE_None if it works or is ignored.

Referenced by GDALDataset::AdviseRead().

43.19.3.41 CPLErr GDALRasterBand::GetHistogram (double *dfMin*, double *dfMax*, int *nBuckets*, int * *panHistogram*, int *bIncludeOutOfRange*, int *bApproxOK*, GDALProgressFunc *pfnProgress*, void * *pProgressData*) [virtual]

Compute raster histogram.

Note that the bucket size is (dfMax-dfMin) / nBuckets.

For example to compute a simple 256 entry histogram of eight bit data, the following would be suitable. The unusual bounds are to ensure that bucket boundaries don't fall right on integer values causing possible errors due to rounding after scaling.

```
int anHistogram[256];

poBand->GetHistogram( -0.5, 255.5, 256, anHistogram, FALSE, FALSE,
                      GDALDummyProgress, NULL );
```

Note that setting bApproxOK will generally result in a subsampling of the file, and will utilize overviews if available. It should generally produce a representative histogram for the data that is suitable for use in generating histogram based luts for instance. Generally bApproxOK is much faster than an exactly computed histogram.

Parameters:

dfMin the lower bound of the histogram.

dfMax the upper bound of the histogram.

nBuckets the number of buckets in panHistogram.

panHistogram array into which the histogram totals are placed.

bIncludeOutOfRange if TRUE values below the histogram range will mapped into panHistogram[0], and values above will be mapped into panHistogram[nBuckets-1] otherwise out of range values are discarded.

bApproxOK TRUE if an approximate, or incomplete histogram OK.

pfnProgress function to report progress to completion.

pProgressData application data to pass to pfnProgress.

Returns:

CE_None on success, or CE_Failure if something goes wrong.

References GDALGetDataTypeSize(), GDT_Byte, GDT_CFloat32, GDT_CFloat64, GDT_CInt16, GDT_CInt32, GDT_Float32, GDT_Float64, GDT_Int16, GDT_Int32, GDT_UInt16, GDT_UInt32, GetHistogram(), GetLockedBlockRef(), GetOverviewCount(), GetRasterSampleOverview(), GetXSize(), GetYSize(), GF_Read, and HasArbitraryOverviews().

Referenced by GDALGetRasterHistogram(), GetDefaultHistogram(), and GetHistogram().

43.19.3.42 CPLErr GDALRasterBand::GetDefaultHistogram (double * *pdfMin*, double * *pdfMax*, int * *pnBuckets*, int ** *ppanHistogram*, int *bForce*, GDALProgressFunc *pfnProgress*, void * *pProgressData*) [virtual]

Fetch default raster histogram.

Note that the bucket size is (dfMax-dfMin) / nBuckets.

For example to compute a simple 256 entry histogram of eight bit data, the following would be suitable. The unusual bounds are to ensure that bucket boundaries don't fall right on integer values causing possible errors due to rounding after scaling.

```
int anHistogram[256];

poBand->GetHistogram( -0.5, 255.5, 256, anHistogram, FALSE, FALSE,
                      GDALDummyProgress, NULL );
```

Note that setting bApproxOK will generally result in a subsampling of the file, and will utilize overviews if available. It should generally produce a representative histogram for the data that is suitable for use in generating histogram based luts for instance. Generally bApproxOK is much faster than an exactly computed histogram.

Parameters:

dfMin the lower bound of the histogram.

dfMax the upper bound of the histogram.

nBuckets the number of buckets in panHistogram.

panHistogram array into which the histogram totals are placed.

bIncludeOutOfRange if TRUE values below the histogram range will mapped into panHistogram[0], and values above will be mapped into panHistogram[nBuckets-1] otherwise out of range values are discarded.

bApproxOK TRUE if an approximate, or incomplete histogram OK.

pfnProgress function to report progress to completion.

pProgressData application data to pass to pfnProgress.

Returns:

CE_None on success, CE_Failure if something goes wrong, or CE_Warning if no default histogram is available.

References GDT_Byte, GetHistogram(), GetRasterDataType(), and GetStatistics().

43.19.3.43 const GDALRasterAttributeTable * GDALRasterBand::GetDefaultRAT () [virtual]

Fetch default Raster Attribute Table.

A RAT will be returned if there is a default one associated with the band, otherwise NULL is returned. The returned RAT is owned by the band and should not be deleted, or altered by the application.

Returns:

NULL, or a pointer to an internal RAT owned by the band.

43.19.3.44 CPLErr GDALRasterBand::SetDefaultRAT (const GDALRasterAttributeTable * *poRAT*) [virtual]

Set default Raster Attribute Table.

Associates a default RAT with the band. If not implemented for the format a CPLE_NotSupported error will be issued. If successful a copy of the RAT is made, the original remains owned by the caller.

Parameters:

poRAT the RAT to assign to the band.

Returns:

CE_None on success or CE_Failure if unsupported or otherwise failing.

The documentation for this class was generated from the following files:

- gdal_priv.h
- gdalrasterband.cpp
- rasterio.cpp

43.20 GDALWarpKernel Class Reference

`#include <gdalwarper.h>`

Public Member Functions

- CPLErr **Validate** ()
- CPLErr **PerformWarp** ()

Public Attributes

- char ** **papszWarpOptions**
- **GDALResampleAlg eResample**
- **GDALDataType eWorkingDataType**
- int **nBands**
- int **nSrcXSize**
- int **nSrcYSize**
- GByte ** **papabySrcImage**
- GUInt32 ** **papanBandSrcValid**
- GUInt32 * **panUnifiedSrcValid**
- float * **pafUnifiedSrcDensity**
- int **nDstXSize**
- int **nDstYSize**
- GByte ** **papabyDstImage**
- GUInt32 * **panDstValid**
- float * **pafDstDensity**
- double **dfXScale**
- double **dfYScale**
- double **dfXFilter**
- double **dfYFilter**
- int **nXRadius**
- int **nYRadius**
- int **nFiltInitX**
- int **nFiltInitY**
- int **nSrcXOff**
- int **nSrcYOff**
- int **nDstXOff**
- int **nDstYOff**
- **GDALTransformerFunc pfnTransformer**
- void * **pTransformerArg**
- GDALProgressFunc **pfnProgress**
- void * **pProgress**
- double **dfProgressBase**
- double **dfProgressScale**

43.20.1 Detailed Description

Low level image warping class.

This class is responsible for low level image warping for one "chunk" of imagery. The class is essentially a structure with all data members public - primarily so that new special-case functions can be added without changing the class declaration.

Applications are normally intended to interactive with warping facilities through the **GDALWarpOperation** (p. **??**) class, though the **GDALWarpKernel** (p. **??**) can in theory be used directly if great care is taken in setting up the control data.

Design Issues My intention is that **PerformWarp()** (p. **??**) would analyse the setup in terms of the datatype, resampling type, and validity/density mask usage and pick one of many specific implementations of the warping algorithm over a continuim of optimization vs. generality. At one end there will be a reference general purpose implementation of the algorithm that supports any data type (working internally in double precision complex), all three resampling types, and any or all of the validity/density masks. At the other end would be highly optimized algorithms for common cases like nearest neighbour resampling on GDT_Byte data with no masks.

The full set of optimized versions have not been decided but we should expect to have at least:

- One for each resampling algorithm for 8bit data with no masks.
- One for each resampling algorithm for float data with no masks.
- One for each resampling algorithm for float data with any/all masks (essentially the generic case for just float data).
- One for each resampling algorithm for 8bit data with support for input validity masks (per band or per pixel). This handles the common case of nodata masking.
- One for each resampling algorithm for float data with support for input validity masks (per band or per pixel). This handles the common case of nodata masking.

Some of the specializations would operate on all bands in one pass (especially the ones without masking would do this), while others might process each band individually to reduce code complexity.

Masking Semantics A detailed explanation of the semantics of the validity and density masks, and their effects on resampling kernels is needed here.

43.20.2 Member Function Documentation

43.20.2.1 CPLErr GDALWarpKernel::Validate ()

Check the settings in the **GDALWarpKernel** (p. **??**), and issue a CPLError() (and return CE_Failure) if the configuration is considered to be invalid for some reason.

This method will also do some standard defaulting such as setting pfnProgress to GDALDummyProgress() if it is NULL.

Returns:

CE_None on success or CE_Failure if an error is detected.

References eResample.

Referenced by PerformWarp().

43.20.2.2 CPLErr GDALWarpKernel::PerformWarp ()

This method performs the warp described in the **GDALWarpKernel** (p. **??**).

Returns:

CE_None on success or CE_Failure if an error occurs.

References eResample, eWorkingDataType, GDT_Byte, GDT_Float32, GDT_Int16, GDT_UInt16, GRA_Bilinear, GRA_Cubic, GRA_CubicSpline, GRA_NearestNeighbour, nDstXSize, nDstYSize, nSrcXSize, nSrcYSize, pafDstDensity, pafUnifiedSrcDensity, panDstValid, panUnifiedSrcValid, papanBandSrcValid, pfnProgress, pProgress, and Validate().

43.20.3 Member Data Documentation

43.20.3.1 GDALResampleAlg GDALWarpKernel::eResample

Resampling algorithm.

The resampling algorithm to use. One of GRA_NearestNeighbour, GRA_Bilinear, or GRA_Cubic.

This field is required. GDT_NearestNeighbour may be used as a default value.

Referenced by PerformWarp(), Validate(), and GDALWarpOperation::WarpRegionToBuffer().

43.20.3.2 GDALDataType GDALWarpKernel::eWorkingDataType

Working pixel data type.

The datatype of pixels in the source image (papabySrcimage) and destination image (papabyDstImage) buffers. Note that operations on some data types (such as GDT_Byte) may be much better optimized than other less common cases.

This field is required. It may not be GDT_Unknown.

Referenced by PerformWarp(), and GDALWarpOperation::WarpRegionToBuffer().

43.20.3.3 int GDALWarpKernel::nBands

Number of bands.

The number of bands (layers) of imagery being warped. Determines the number of entries in the papabySrcImage, papanBandSrcValid, and papabyDstImage arrays.

This field is required.

Referenced by GDALWarpOperation::WarpRegionToBuffer().

43.20.3.4 int GDALWarpKernel::nSrcXSize

Source image width in pixels.

This field is required.

Referenced by PerformWarp(), and GDALWarpOperation::WarpRegionToBuffer().

43.20.3.5 int GDALWarpKernel::nSrcYSize

Source image height in pixels.

This field is required.

Referenced by PerformWarp(), and GDALWarpOperation::WarpRegionToBuffer().

43.20.3.6 int GDALWarpKernel::papabySrcImage

Array of source image band data.

This is an array of pointers (of size **GDALWarpKernel::nBands** (p. **??**)) pointers to image data. Each individual band of image data is organized as a single block of image data in left to right, then bottom to top order. The actual type of the image data is determined by **GDALWarpKernel::eWorkingDataType** (p. **??**).

To access the the pixel value for the (x=3,y=4) pixel (zero based) of the second band with eWorkingDataType set to GDT_Float32 use code like this:

```
float dfPixelValue;
int   nBand = 1;  // band indexes are zero based.
int   nPixel = 3; // zero based
int   nLine = 4;  // zero based

assert( nPixel >= 0 && nPixel < poKern->nSrcXSize );
assert( nLine >= 0 && nLine < poKern->nSrcYSize );
assert( nBand >= 0 && nBand < poKern->nBands );
dfPixelValue = ((float *) poKern->papabySrcImage[nBand-1])
                              [nPixel + nLine * poKern->nSrcXSize];
```

This field is required.

Referenced by GDALWarpOperation::WarpRegionToBuffer().

43.20.3.7 GUInt32 ** GDALWarpKernel::papanBandSrcValid

Per band validity mask for source pixels.

Array of pixel validity mask layers for each source band. Each of the mask layers is the same size (in pixels) as the source image with one bit per pixel. Note that it is legal (and common) for this to be NULL indicating that none of the pixels are invalidated, or for some band validity masks to be NULL in which case all pixels of the band are valid. The following code can be used to test the validity of a particular pixel.

```
int   bIsValid = TRUE;
int   nBand = 1;  // band indexes are zero based.
int   nPixel = 3; // zero based
int   nLine = 4;  // zero based

assert( nPixel >= 0 && nPixel < poKern->nSrcXSize );
assert( nLine >= 0 && nLine < poKern->nSrcYSize );
assert( nBand >= 0 && nBand < poKern->nBands );

if( poKern->papanBandSrcValid != NULL
    && poKern->papanBandSrcValid[nBand] != NULL )
```

```
{
    GUInt32 *panBandMask = poKern->papanBandSrcValid[nBand];
    int    iPixelOffset = nPixel + nLine * poKern->nSrcXSize;

    bIsValid = panBandMask[iPixelOffset>>5]
               & (0x01 << (iPixelOffset & 0x1f));
}
```

Referenced by PerformWarp(), and GDALWarpOperation::WarpRegionToBuffer().

43.20.3.8 GUInt32 * GDALWarpKernel::panUnifiedSrcValid

Per pixel validity mask for source pixels.

A single validity mask layer that applies to the pixels of all source bands. It is accessed similarly to papanBandSrcValid, but without the extra level of band indirection.

This pointer may be NULL indicating that all pixels are valid.

Note that if both panUnifiedSrcValid, and papanBandSrcValid are available, the pixel isn't considered to be valid unless both arrays indicate it is valid.

Referenced by PerformWarp(), and GDALWarpOperation::WarpRegionToBuffer().

43.20.3.9 float * GDALWarpKernel::pafUnifiedSrcDensity

Per pixel density mask for source pixels.

A single density mask layer that applies to the pixels of all source bands. It contains values between 0.0 and 1.0 indicating the degree to which this pixel should be allowed to contribute to the output result.

This pointer may be NULL indicating that all pixels have a density of 1.0.

The density for a pixel may be accessed like this:

```
float fDensity = 1.0;
int   nPixel = 3; // zero based
int   nLine = 4;  // zero based

assert( nPixel >= 0 && nPixel < poKern->nSrcXSize );
assert( nLine >= 0 && nLine < poKern->nSrcYSize );
if( poKern->pafUnifiedSrcDensity != NULL )
  fDensity = poKern->pafUnifiedSrcDensity
                               [nPixel + nLine * poKern->nSrcXSize];
```

Referenced by PerformWarp(), and GDALWarpOperation::WarpRegionToBuffer().

43.20.3.10 int GDALWarpKernel::nDstXSize

Width of destination image in pixels.

This field is required.

Referenced by PerformWarp(), and GDALWarpOperation::WarpRegionToBuffer().

43.20.3.11 int GDALWarpKernel::nDstYSize

Height of destination image in pixels.

This field is required.

Referenced by PerformWarp(), and GDALWarpOperation::WarpRegionToBuffer().

43.20.3.12 GByte ** GDALWarpKernel::papabyDstImage

Array of destination image band data.

This is an array of pointers (of size **GDALWarpKernel::nBands** (p. **??**)) pointers to image data. Each individual band of image data is organized as a single block of image data in left to right, then bottom to top order. The actual type of the image data is determined by **GDALWarpKernel::eWorkingDataType** (p. **??**).

To access the the pixel value for the (x=3,y=4) pixel (zero based) of the second band with eWorkingDataType set to GDT_Float32 use code like this:

```
float dfPixelValue;
int   nBand = 1;  // band indexes are zero based.
int   nPixel = 3; // zero based
int   nLine = 4;  // zero based

assert( nPixel >= 0 && nPixel < poKern->nDstXSize );
assert( nLine >= 0 && nLine < poKern->nDstYSize );
assert( nBand >= 0 && nBand < poKern->nBands );
dfPixelValue = ((float *) poKern->papabyDstImage[nBand-1])
                              [nPixel + nLine * poKern->nSrcYSize];
```

This field is required.

Referenced by GDALWarpOperation::WarpRegionToBuffer().

43.20.3.13 GUInt32 * GDALWarpKernel::panDstValid

Per pixel validity mask for destination pixels.

A single validity mask layer that applies to the pixels of all destination bands. It is accessed similarly to papanUnitifiedSrcValid, but based on the size of the destination image.

This pointer may be NULL indicating that all pixels are valid.

Referenced by PerformWarp(), and GDALWarpOperation::WarpRegionToBuffer().

43.20.3.14 float * GDALWarpKernel::pafDstDensity

Per pixel density mask for destination pixels.

A single density mask layer that applies to the pixels of all destination bands. It contains values between 0.0 and 1.0.

This pointer may be NULL indicating that all pixels have a density of 1.0.

The density for a pixel may be accessed like this:

```
float fDensity = 1.0;
int   nPixel = 3; // zero based
int   nLine = 4;  // zero based

assert( nPixel >= 0 && nPixel < poKern->nDstXSize );
assert( nLine >= 0 && nLine < poKern->nDstYSize );
if( poKern->pafDstDensity != NULL )
  fDensity = poKern->pafDstDensity[nPixel + nLine * poKern->nDstXSize];
```

Referenced by PerformWarp(), and GDALWarpOperation::WarpRegionToBuffer().

43.20.3.15 int GDALWarpKernel::nSrcXOff

X offset to source pixel coordinates for transformation.

See pfnTransformer.

This field is required.

Referenced by GDALWarpOperation::WarpRegionToBuffer().

43.20.3.16 int GDALWarpKernel::nSrcYOff

Y offset to source pixel coordinates for transformation.

See pfnTransformer.

This field is required.

Referenced by GDALWarpOperation::WarpRegionToBuffer().

43.20.3.17 int GDALWarpKernel::nDstXOff

X offset to destination pixel coordinates for transformation.

See pfnTransformer.

This field is required.

Referenced by GDALWarpOperation::WarpRegionToBuffer().

43.20.3.18 int GDALWarpKernel::nDstYOff

Y offset to destination pixel coordinates for transformation.

See pfnTransformer.

This field is required.

Referenced by GDALWarpOperation::WarpRegionToBuffer().

43.20.3.19 GDALTransformerFunc GDALWarpKernel::pfnTransformer

Source/destination location transformer.

The function to call to transform coordinates between source image pixel/line coordinates and destination image pixel/line coordinates. See **GDALTransformerFunc()** (p. **??**) for details of the semantics of this function.

The GDALWarpKern algorithm will only ever use this transformer in "destination to source" mode (bDstToSrc=TRUE), and will always pass partial or complete scanlines of points in the destination image as input. This means, amoung other things, that it is safe to the the approximating transform **GDALApproxTransform()** (p. **??**) as the transformation function.

Source and destination images may be subsets of a larger overall image. The transformation algorithms will expect and return pixel/line coordinates in terms of this larger image, so coordinates need to be offset by

the offsets specified in nSrcXOff, nSrcYOff, nDstXOff, and nDstYOff before passing to pfnTransformer, and after return from it.

The GDALWarpKernel::pfnTransformerArg value will be passed as the callback data to this function when it is called.

This field is required.

Referenced by GDALWarpOperation::WarpRegionToBuffer().

43.20.3.20 void * GDALWarpKernel::pTransformerArg

Callback data for pfnTransformer.

This field may be NULL if not required for the pfnTransformer being used.

Referenced by GDALWarpOperation::WarpRegionToBuffer().

43.20.3.21 GDALProgressFunc GDALWarpKernel::pfnProgress

The function to call to report progress of the algorithm, and to check for a requested termination of the operation. It operates according to GDALProgressFunc() semantics.

Generally speaking the progress function will be invoked for each scanline of the destination buffer that has been processed.

This field may be NULL (internally set to GDALDummyProgress()).

Referenced by PerformWarp(), and GDALWarpOperation::WarpRegionToBuffer().

43.20.3.22 void * GDALWarpKernel::pProgress

Callback data for pfnProgress.

This field may be NULL if not required for the pfnProgress being used.

Referenced by PerformWarp(), and GDALWarpOperation::WarpRegionToBuffer().

The documentation for this class was generated from the following files:

- **gdalwarper.h**
- gdalwarpkernel.cpp

43.21 GDALWarpOperation Class Reference

`#include <gdalwarper.h>`

Public Member Functions

- CPLErr **Initialize** (const **GDALWarpOptions** *psNewOptions)
- const **GDALWarpOptions** * **GetOptions** ()
- CPLErr **ChunkAndWarpImage** (int nDstXOff, int nDstYOff, int nDstXSize, int nDstYSize)
- CPLErr **ChunkAndWarpMulti** (int nDstXOff, int nDstYOff, int nDstXSize, int nDstYSize)
- CPLErr **WarpRegion** (int nDstXOff, int nDstYOff, int nDstXSize, int nDstYSize, int nSrcXOff=0, int nSrcYOff=0, int nSrcXSize=0, int nSrcYSize=0)
- CPLErr **WarpRegionToBuffer** (int nDstXOff, int nDstYOff, int nDstXSize, int nDstYSize, void *pDataBuf, **GDALDataType** eBufDataType, int nSrcXOff=0, int nSrcYOff=0, int nSrcXSize=0, int nSrcYSize=0)

43.21.1 Detailed Description

High level image warping class.

Warper Design

The overall GDAL high performance image warper is split into a few components.

- The transformation between input and output file coordinates is handled via **GDALTransformerFunc()** (p. **??**) implementations such as the one returned by **GDALCreateGenImgProjTransformer()** (p. **??**). The transformers are ultimately responsible for translating pixel/line locations on the destination image to pixel/line locations on the source image.

- In order to handle images too large to hold in RAM, the warper needs to segment large images. This is the responsibility of the **GDALWarpOperation** (p. **??**) class. The **GDALWarpOperation::ChunkAndWarpImage()** (p. **??**) invokes **GDALWarpOperation::WarpRegion()** (p. **??**) on chunks of output and input image that are small enough to hold in the amount of memory allowed by the application. This process is described in greater detail in the **Image Chunking** section.

- The **GDALWarpOperation::WarpRegion()** (p. **??**) function creates and loads an output image buffer, and then calls **WarpRegionToBuffer()** (p. **??**).

- **GDALWarpOperation::WarpRegionToBuffer()** (p. **??**) is responsible for loading the source imagery corresponding to a particular output region, and generating masks and density masks from the source and destination imagery using the generator functions found in the **GDALWarpOptions** (p. **??**) structure. Binds this all into an instance of **GDALWarpKernel** (p. **??**) on which the **GDALWarpKernel::PerformWarp()** (p. **??**) method is called.

- **GDALWarpKernel** (p. **??**) does the actual image warping, but is given an input image and an output image to operate on. The **GDALWarpKernel** (p. **??**) does no IO, and in fact knows nothing about GDAL. It invokes the transformation function to get sample locations, builds output values based on the resampling algorithm in use. It also takes any validity and density masks into account during this operation.

Chunk Size Selection The **GDALWarpOptions** (p. **??**) **ChunkAndWarpImage()** (p. **??**) method is responsible for invoking the **WarpRegion()** (p. **??**) method on appropriate sized output chunks such that the memory required for the output image buffer, input image buffer and any required density and validity buffers is less than or equal to the application defined maximum memory available for use.

It checks the memory requrired by walking the edges of the output region, transforming the locations back into source pixel/line coordinates and establishing a bounding rectangle of source imagery that would be required for the output area. This is actually accomplished by the private GDALWarpOperation::ComputeSourceWindow() method.

Then memory requirements are used by totaling the memory required for all output bands, input bands, validity masks and density masks. If this is greater than the **GDALWarpOptions::dfWarpMemoryLimit** (p. **??**) then the destination region is divided in two (splitting the longest dimension), and **ChunkAndWarpImage()** (p. **??**) recursively invoked on each destination subregion.

Validity and Density Masks Generation Fill in ways in which the validity and density masks may be generated here. Note that detailed semantics of the masks should be found in **GDALWarpKernel** (p. **??**).

43.21.2 Member Function Documentation

43.21.2.1 CPLErr GDALWarpOperation::Initialize (const GDALWarpOptions * *psNewOptions*)

This method initializes the GDALWarpOperation's concept of the warp options in effect. It creates an internal copy of the **GDALWarpOptions** (p. **??**) structure and defaults a variety of additional fields in the internal copy if not set in the provides warp options.

Defaulting operations include:

- If the nBandCount is 0, it will be set to the number of bands in the source image (which must match the output image) and the panSrcBands and panDstBands will be populated.

Parameters:

psNewOptions input set of warp options. These are copied and may be destroyed after this call by the application.

Returns:

CE_None on success or CE_Failure if an error occurs.

References GDALWarpOptions::dfCutlineBlendDist, GDALWarpOptions::dfWarpMemoryLimit, GDALWarpOptions::eWorkingDataType, GDALDataTypeIsComplex(), GDALDataTypeUnion(), GDALGetRasterBand(), GDALGetRasterCount(), GDALGetRasterDataType(), GDT_Byte, GDT_Int16, GDT_Int32, GDT_UInt16, GDT_UInt32, GDALWarpOptions::hCutline, GDALWarpOptions::hDstDS, GDALWarpOptions::hSrcDS, GDALWarpOptions::nBandCount, GDALWarpOptions::padfSrcNoDataImag, GDALWarpOptions::padfSrcNoDataReal, GDALWarpOptions::panDstBands, GDALWarpOptions::panSrcBands, and GDALWarpOptions::papszWarpOptions.

Referenced by GDALCreateWarpOperation(), and GDALReprojectImage().

43.21.2.2 CPLErr GDALWarpOperation::ChunkAndWarpImage (int *nDstXOff*, int *nDstYOff*, int *nDstXSize*, int *nDstYSize*)

This method does a complete warp of the source image to the destination image for the indicated region with the current warp options in effect. Progress is reported to the installed progress monitor, if any.

This function will subdivide the region and recursively call itself until the total memory required to process a region chunk will all fit in the memory pool defined by **GDALWarpOptions::dfWarpMemoryLimit** (p. **??**).

Once an appropriate region is selected **GDALWarpOperation::WarpRegion()** (p. **??**) is invoked to do the actual work.

Parameters:

nDstXOff X offset to window of destination data to be produced.

nDstYOff Y offset to window of destination data to be produced.

nDstXSize Width of output window on destination file to be produced.

nDstYSize Height of output window on destination file to be produced.

Returns:

CE_None on success or CE_Failure if an error occurs.

References GDALWarpOptions::pfnProgress, GDALWarpOptions::pProgressArg, and WarpRegion().

Referenced by GDALChunkAndWarpImage(), and GDALReprojectImage().

43.21.2.3 CPLErr GDALWarpOperation::ChunkAndWarpMulti (int *nDstXOff*, int *nDstYOff*, int *nDstXSize*, int *nDstYSize*)

This method does a complete warp of the source image to the destination image for the indicated region with the current warp options in effect. Progress is reported to the installed progress monitor, if any.

Externally this method operates the same as **ChunkAndWarpImage()** (p. **??**), but internally this method uses multiple threads to interleave input/output for one region while the processing is being done for another.

Parameters:

nDstXOff X offset to window of destination data to be produced.

nDstYOff Y offset to window of destination data to be produced.

nDstXSize Width of output window on destination file to be produced.

nDstYSize Height of output window on destination file to be produced.

Returns:

CE_None on success or CE_Failure if an error occurs.

Referenced by GDALChunkAndWarpMulti().

43.21.2.4 CPLErr GDALWarpOperation::WarpRegion (int *nDstXOff*, int *nDstYOff*, int *nDstXSize*, int *nDstYSize*, int *nSrcXOff* = 0, int *nSrcYOff* = 0, int *nSrcXSize* = 0, int *nSrcYSize* = 0)

This method requests the indicated region of the output file be generated.

Note that **WarpRegion()** (p. **??**) will produce the requested area in one low level warp operation without verifying that this does not exceed the stated memory limits for the warp operation. Applications should take care not to call **WarpRegion()** (p. **??**) on too large a region! This function is normally called by

ChunkAndWarpImage() (p. **??**), the normal entry point for applications. Use it instead if staying within memory constraints is desired.

Progress is reported from 0.0 to 1.0 for the indicated region.

Parameters:

nDstXOff X offset to window of destination data to be produced.

nDstYOff Y offset to window of destination data to be produced.

nDstXSize Width of output window on destination file to be produced.

nDstYSize Height of output window on destination file to be produced.

Returns:

CE_None on success or CE_Failure if an error occurs.

References GDALWarpOptions::eWorkingDataType, GDALCopyWords(), GDALDatasetRasterIO(), GDALFlushCache(), GDALGetDataTypeSize(), GDT_Byte, GDT_CFloat64, GDT_Float64, GF_Read, GF_Write, GDALWarpOptions::hDstDS, GDALWarpOptions::nBandCount, GDALWarpOptions::padfDstNoDataImag, GDALWarpOptions::padfDstNoDataReal, GDALWarpOptions::panDstBands, GDALWarpOptions::papszWarpOptions, and WarpRegionToBuffer().

Referenced by ChunkAndWarpImage(), and GDALWarpRegion().

43.21.2.5 CPLErr GDALWarpOperation::WarpRegionToBuffer (int *nDstXOff*, int *nDstYOff*, int *nDstXSize*, int *nDstYSize*, void * *pDataBuf*, GDALDataType *eBufDataType*, int *nSrcXOff* = 0, int *nSrcYOff* = 0, int *nSrcXSize* = 0, int *nSrcYSize* = 0)

This method requests that a particular window of the output dataset be warped and the result put into the provided data buffer. The output dataset doesn't even really have to exist to use this method as long as the transformation function in the **GDALWarpOptions** (p. **??**) is setup to map to a virtual pixel/line space.

This method will do the whole region in one chunk, so be wary of the amount of memory that might be used.

Parameters:

nDstXOff X offset to window of destination data to be produced.

nDstYOff Y offset to window of destination data to be produced.

nDstXSize Width of output window on destination file to be produced.

nDstYSize Height of output window on destination file to be produced.

pDataBuf the data buffer to place result in, of type eBufDataType.

eBufDataType the type of the output data buffer. For now this must match **GDALWarpOptions::eWorkingDataType** (p. **??**).

nSrcXOff source window X offset (computed if window all zero)

nSrcYOff source window Y offset (computed if window all zero)

nSrcXSize source window X size (computed if window all zero)

nSrcYSize source window Y size (computed if window all zero)

Returns:

CE_None on success or CE_Failure if an error occurs.

References GDALWarpKernel::dfProgressBase, GDALWarpKernel::dfProgressScale, GDALWarpKernel::eResample, GDALWarpOptions::eResampleAlg, GDALWarpKernel::eWorkingDataType, GDALWarpOptions::eWorkingDataType, GDALDatasetRasterIO(), GDALGetDataTypeSize(), GF_Read, GDALWarpOptions::hCutline, GDALWarpOptions::hSrcDS, GDALWarpOptions::nBandCount, GDALWarpKernel::nBands, GDALWarpOptions::nDstAlphaBand, GDALWarpKernel::nDstXOff, GDALWarpKernel::nDstXSize, GDALWarpKernel::nDstYOff, GDALWarpKernel::nDstYSize, GDALWarpOptions::nSrcAlphaBand, GDALWarpKernel::nSrcXOff, GDALWarpKernel::nSrcXSize, GDALWarpKernel::nSrcYOff, GDALWarpKernel::nSrcYSize, GDALWarpOptions::padfDstNoDataImag, GDALWarpOptions::padfDstNoDataReal, GDALWarpOptions::padfSrcNoDataImag, GDALWarpOptions::padfSrcNoDataReal, GDALWarpKernel::pafDstDensity, GDALWarpKernel::pafUnifiedSrcDensity, GDALWarpKernel::panDstValid, GDALWarpOptions::panSrcBands, GDALWarpKernel::panUnifiedSrcValid, GDALWarpKernel::papabyDstImage, GDALWarpKernel::papabySrcImage, GDALWarpKernel::papanBandSrcValid, GDALWarpOptions::papszWarpOptions, GDALWarpKernel::papszWarpOptions, GDALWarpOptions::pfnPostWarpChunkProcessor, GDALWarpOptions::pfnPreWarpChunkProcessor, GDALWarpOptions::pfnProgress, GDALWarpKernel::pfnProgress, GDALWarpOptions::pfnTransformer, GDALWarpKernel::pfnTransformer, GDALWarpOptions::pPostWarpProcessorArg, GDALWarpOptions::pPreWarpProcessorArg, GDALWarpKernel::pProgress, GDALWarpOptions::pProgressArg, GDALWarpOptions::pTransformerArg, and GDALWarpKernel::pTransformerArg.

Referenced by GDALWarpRegionToBuffer(), and WarpRegion().

The documentation for this class was generated from the following files:

- **gdalwarper.h**
- gdalwarpoperation.cpp

43.22 GDALWarpOptions Struct Reference

`#include <gdalwarper.h>`

Public Attributes

- char ** **papszWarpOptions**
- double **dfWarpMemoryLimit**
- **GDALResampleAlg eResampleAlg**
- **GDALDataType eWorkingDataType**
- GDALDatasetH **hSrcDS**
- GDALDatasetH **hDstDS**
- int **nBandCount**
- int * **panSrcBands**
- int * **panDstBands**
- int **nSrcAlphaBand**
- int **nDstAlphaBand**
- double * **padfSrcNoDataReal**
- double * **padfSrcNoDataImag**
- double * **padfDstNoDataReal**
- double * **padfDstNoDataImag**
- GDALProgressFunc **pfnProgress**
- void * **pProgressArg**
- **GDALTransformerFunc pfnTransformer**
- void * **pTransformerArg**
- GDALMaskFunc * **papfnSrcPerBandValidityMaskFunc**
- void ** **papSrcPerBandValidityMaskFuncArg**
- GDALMaskFunc **pfnSrcValidityMaskFunc**
- void * **pSrcValidityMaskFuncArg**
- GDALMaskFunc **pfnSrcDensityMaskFunc**
- void * **pSrcDensityMaskFuncArg**
- GDALMaskFunc **pfnDstDensityMaskFunc**
- void * **pDstDensityMaskFuncArg**
- GDALMaskFunc **pfnDstValidityMaskFunc**
- void * **pDstValidityMaskFuncArg**
- CPLErr(* **pfnPreWarpChunkProcessor**)(void *pKern, void *pArg)
- void * **pPreWarpProcessorArg**
- CPLErr(* **pfnPostWarpChunkProcessor**)(void *pKern, void *pArg)
- void * **pPostWarpProcessorArg**
- void * **hCutline**
- double **dfCutlineBlendDist**

43.22.1 Detailed Description

Warp control options for use with **GDALWarpOperation::Initialize()** (p. **??**)

43.22.2 Member Data Documentation

43.22.2.1 char ∗∗ GDALWarpOptions::papszWarpOptions

A string list of additional options controlling the warp operation in name=value format. A suitable string list can be prepared with **CSLSetNameValue()** (p. **??**).

The following values are currently supported:

- 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 overlayed.

- 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 alot of extra seeking around the disk, and reduced IO throughput. The default at this time is NO.

- SKIP_NOSOURCE=YES/NO: Skip all processing for chunks for which there is no corresponding input data. This will disable initializing the destination (INIT_DEST) and all other processing, and so should be used careful. Mostly useful to short circuit a lot of extra work in mosaicing situations.

Normally when computing the source raster data to load to generate a particular output area, the warper samples transforms 21 points along each edge of the destination region back onto the source file, and uses this to compute a bounding window on the source image that is sufficient. Depending on the transformation in effect, the source window may be a bit too small, or even missing large areas. Problem situations are those where the transformation is very non-linear or "inside out". Examples are transforming from WGS84 to Polar Steregraphic for areas around the pole, or transformations where some of the image is untransformable. The following options provide some additional control to deal with errors in computing the source window:

- SAMPLE_GRID=YES/NO: Setting this option to YES will force the sampling to include internal points as well as edge points which can be important if the transformation is esoteric inside out, or if large sections of the destination image are not transformable into the source coordinate system.

- SAMPLE_STEPS: Modifies the density of the sampling grid. The default number of steps is 21. Increasing this can increase the computational cost, but improves the accuracy with which the source region is computed.

- SOURCE_EXTRA: This is a number of extra pixels added around the source window for a given request, and by default it is 1 to take care of rounding error. Setting this larger will incease the amount of data that needs to be read, but can avoid missing source data.

Referenced by GDALWarpOperation::Initialize(), GDALWarpOperation::WarpRegion(), and GDALWarp-Operation::WarpRegionToBuffer().

43.22.2.2 double GDALWarpOptions::dfWarpMemoryLimit

In bytes, 0.0 for internal default

Referenced by GDALWarpOperation::Initialize().

43.22.2.3 GDALResampleAlg GDALWarpOptions::eResampleAlg

Resampling algorithm to use

Referenced by GDALAutoCreateWarpedVRT(), GDALReprojectImage(), and GDALWarpOperation::WarpRegionToBuffer().

43.22.2.4 GDALDataType GDALWarpOptions::eWorkingDataType

data type to use during warp operation, GDT_Unknown lets the algorithm select the type

Referenced by GDALWarpOperation::Initialize(), GDALWarpOperation::WarpRegion(), and GDALWarpOperation::WarpRegionToBuffer().

43.22.2.5 GDALDatasetH GDALWarpOptions::hSrcDS

Source image dataset.

Referenced by GDALAutoCreateWarpedVRT(), GDALReprojectImage(), GDALWarpOperation::Initialize(), and GDALWarpOperation::WarpRegionToBuffer().

43.22.2.6 GDALDatasetH GDALWarpOptions::hDstDS

Destination image dataset - may be NULL if only using **GDALWarpOperation::WarpRegionToBuffer()** (p. **??**).

Referenced by GDALCreateWarpedVRT(), GDALReprojectImage(), GDALWarpOperation::Initialize(), and GDALWarpOperation::WarpRegion().

43.22.2.7 int GDALWarpOptions::nBandCount

Number of bands to process, may be 0 to select all bands.

Referenced by GDALAutoCreateWarpedVRT(), GDALCreateWarpedVRT(), GDALReprojectImage(), GDALWarpOperation::Initialize(), GDALWarpOperation::WarpRegion(), and GDALWarpOperation::WarpRegionToBuffer().

43.22.2.8 int* GDALWarpOptions::panSrcBands

The band numbers for the source bands to process (1 based)

Referenced by GDALAutoCreateWarpedVRT(), GDALReprojectImage(), GDALWarpOperation::Initialize(), and GDALWarpOperation::WarpRegionToBuffer().

43.22.2.9 int* GDALWarpOptions::panDstBands

The band numbers for the destination bands to process (1 based)

Referenced by GDALAutoCreateWarpedVRT(), GDALReprojectImage(), GDALWarpOperation::Initialize(), and GDALWarpOperation::WarpRegion().

43.22.2.10 int GDALWarpOptions::nSrcAlphaBand

The source band so use as an alpha (transparency) value, 0=disabled

Referenced by GDALWarpOperation::WarpRegionToBuffer().

43.22.2.11 int GDALWarpOptions::nDstAlphaBand

The dest. band so use as an alpha (transparency) value, 0=disabled

Referenced by GDALWarpOperation::WarpRegionToBuffer().

43.22.2.12 double* GDALWarpOptions::padfSrcNoDataReal

The "nodata" value real component for each input band, if NULL there isn't one

Referenced by GDALReprojectImage(), GDALWarpOperation::Initialize(), and GDALWarpOperation::WarpRegionToBuffer().

43.22.2.13 double* GDALWarpOptions::padfSrcNoDataImag

The "nodata" value imaginary component - may be NULL even if real component is provided.

Referenced by GDALReprojectImage(), GDALWarpOperation::Initialize(), and GDALWarpOperation::WarpRegionToBuffer().

43.22.2.14 double* GDALWarpOptions::padfDstNoDataReal

The "nodata" value real component for each output band, if NULL there isn't one

Referenced by GDALWarpOperation::WarpRegion(), and GDALWarpOperation::WarpRegionToBuffer().

43.22.2.15 double* GDALWarpOptions::padfDstNoDataImag

The "nodata" value imaginary component - may be NULL even if real component is provided.

Referenced by GDALWarpOperation::WarpRegion(), and GDALWarpOperation::WarpRegionToBuffer().

43.22.2.16 GDALProgressFunc GDALWarpOptions::pfnProgress

GDALProgressFunc() compatible progress reporting function, or NULL if there isn't one.

Referenced by GDALWarpOperation::ChunkAndWarpImage(), GDALReprojectImage(), and GDALWarpOperation::WarpRegionToBuffer().

43.22.2.17 void* GDALWarpOptions::pProgressArg

Callback argument to be passed to pfnProgress.

Referenced by GDALWarpOperation::ChunkAndWarpImage(), GDALReprojectImage(), and GDALWarpOperation::WarpRegionToBuffer().

43.22.2.18 GDALTransformerFunc GDALWarpOptions::pfnTransformer

Type of spatial point transformer function

Referenced by GDALAutoCreateWarpedVRT(), GDALReprojectImage(), and GDALWarpOperation::WarpRegionToBuffer().

43.22.2.19 void* GDALWarpOptions::pTransformerArg

Handle to image transformer setup structure

Referenced by GDALAutoCreateWarpedVRT(), GDALReprojectImage(), and GDALWarpOperation::WarpRegionToBuffer().

The documentation for this struct was generated from the following files:

- **gdalwarper.h**
- gdalwarper.cpp

Chapter 44

File Documentation

44.1 cpl_conv.h File Reference

```
#include "cpl_port.h"
#include "cpl_vsi.h"
#include "cpl_error.h"
```

Classes

- struct **CPLSharedFileInfo**
- class **CPLLocaleC**

Defines

- #define **CPLFree** VSIFree

Typedefs

- typedef const char *(* **CPLFileFinder**)(const char *, const char *)

Functions

- void **CPLVerifyConfiguration** (void)
- const char * **CPLGetConfigOption** (const char *, const char *)
- void **CPLSetConfigOption** (const char *, const char *)
- void **CPLFreeConfig** (void)
- void * **CPLMalloc** (size_t)
- void * **CPLCalloc** (size_t, size_t)
- void * **CPLRealloc** (void *, size_t)
- char * **CPLStrdup** (const char *)
- char * **CPLStrlwr** (char *)
- char * **CPLFGets** (char *, int, FILE *)
- const char * **CPLReadLine** (FILE *)

- const char * **CPLReadLineL** (FILE *)
- double **CPLAtof** (const char *)
- double **CPLAtofDelim** (const char *, char)
- double **CPLStrtod** (const char *, char **)
- double **CPLStrtodDelim** (const char *, char **, char)
- float **CPLStrtof** (const char *, char **)
- float **CPLStrtofDelim** (const char *, char **, char)
- double **CPLAtofM** (const char *)
- char * **CPLScanString** (const char *, int, int, int)
- double **CPLScanDouble** (const char *, int)
- long **CPLScanLong** (const char *, int)
- unsigned long **CPLScanULong** (const char *, int)
- GUIntBig **CPLScanUIntBig** (const char *, int)
- void * **CPLScanPointer** (const char *, int)
- int **CPLPrintString** (char *, const char *, int)
- int **CPLPrintStringFill** (char *, const char *, int)
- int **CPLPrintInt32** (char *, GInt32, int)
- int **CPLPrintUIntBig** (char *, GUIntBig, int)
- int **CPLPrintDouble** (char *, const char *, double, const char *)
- int **CPLPrintTime** (char *, int, const char *, const struct tm *, const char *)
- int **CPLPrintPointer** (char *, void *, int)
- void * **CPLGetSymbol** (const char *, const char *)
- int **CPLGetExecPath** (char *pszPathBuf, int nMaxLength)
- const char * **CPLGetPath** (const char *)
- const char * **CPLGetDirname** (const char *)
- const char * **CPLGetFilename** (const char *)
- const char * **CPLGetBasename** (const char *)
- const char * **CPLGetExtension** (const char *)
- char * **CPLGetCurrentDir** (void)
- const char * **CPLFormFilename** (const char *pszPath, const char *pszBasename, const char *pszExtension)
- const char * **CPLFormCIFilename** (const char *pszPath, const char *pszBasename, const char *pszExtension)
- const char * **CPLResetExtension** (const char *, const char *)
- const char * **CPLProjectRelativeFilename** (const char *pszProjectDir, const char *pszSecondaryFilename)
- int **CPLIsFilenameRelative** (const char *pszFilename)
- const char * **CPLExtractRelativePath** (const char *, const char *, int *)
- const char * **CPLCleanTrailingSlash** (const char *)
- char ** **CPLCorrespondingPaths** (const char *pszOldFilename, const char *pszNewFilename, char **papszFileList)
- int **CPLCheckForFile** (char *pszFilename, char **papszSiblingList)
- const char * **CPLGenerateTempFilename** (const char *pszStem)
- const char * **CPLFindFile** (const char *pszClass, const char *pszBasename)
- const char * **CPLDefaultFindFile** (const char *pszClass, const char *pszBasename)
- void **CPLPushFileFinder** (CPLFileFinder pfnFinder)
- CPLFileFinder **CPLPopFileFinder** (void)
- void **CPLPushFinderLocation** (const char *)
- void **CPLPopFinderLocation** (void)
- void **CPLFinderClean** (void)

- int **CPLStat** (const char *, VSIStatBuf *)
- FILE * **CPLOpenShared** (const char *, const char *, int)
- void **CPLCloseShared** (FILE *)
- CPLSharedFileInfo * **CPLGetSharedList** (int *)
- void **CPLDumpSharedList** (FILE *)
- double **CPLDMSToDec** (const char *is)
- const char * **CPLDecToDMS** (double dfAngle, const char *pszAxis, int nPrecision)
- double **CPLPackedDMSToDec** (double)
- double **CPLDecToPackedDMS** (double dfDec)
- void **CPLStringToComplex** (const char *pszString, double *pdfReal, double *pdfImag)
- int **CPLUnlinkTree** (const char *)
- int **CPLCopyFile** (const char *pszNewPath, const char *pszOldPath)
- int **CPLMoveFile** (const char *pszNewPath, const char *pszOldPath)

44.1.1 Detailed Description

Various convenience functions for CPL.

44.1.2 Function Documentation

44.1.2.1 double CPLAtof (const char * *nptr*)

Converts ASCII string to floating point number.

This function converts the initial portion of the string pointed to by nptr to double floating point representation. The behaviour is the same as

CPLStrtod(nptr, (char **)NULL);

This function does the same as standard atof(3), but does not take locale in account. That means, the decimal delimiter is always '.' (decimal point). Use **CPLAtofDelim()** (p. **??**) function if you want to specify custom delimiter.

IMPORTANT NOTE. Existance of this function does not mean you should always use it. Sometimes you should use standard locale aware atof(3) and its family. When you need to process the user's input (for example, command line parameters) use atof(3), because user works in localized environment and her input will be done accordingly the locale set. In particular that means we should not make assumptions about character used as decimal delimiter, it can be either "." or ",". But when you are parsing some ASCII file in predefined format, you most likely need **CPLAtof()** (p. **??**), because such files distributed across the systems with different locales and floating point representation shoudl be considered as a part of file format. If the format uses "." as a delimiter the same character must be used when parsing number regardless of actual locale setting.

Parameters:

nptr Pointer to string to convert.

Returns:

Converted value, if any.

References CPLAtof(), and CPLStrtod().

Referenced by CPLAtof(), CPLScanDouble(), and GDALCreateRPCTransformer().

44.1.2.2 double CPLAtofDelim (const char * *nptr*, char *point*)

Converts ASCII string to floating point number.

This function converts the initial portion of the string pointed to by nptr to double floating point representation. The behaviour is the same as

CPLStrtodDelim(nptr, (char **)NULL, point);

This function does the same as standard atof(3), but does not take locale in account. Instead of locale defined decimal delimiter you can specify your own one. Also see notes for **CPLAtof()** (p. **??**) function.

Parameters:

nptr Pointer to string to convert.

point Decimal delimiter.

Returns:

Converted value, if any.

References CPLAtofDelim(), and CPLStrtodDelim().

Referenced by CPLAtofDelim().

44.1.2.3 double CPLAtofM (const char * *nptr*)

Converts ASCII string to floating point number using any numeric locale.

This function converts the initial portion of the string pointed to by nptr to double floating point representation. This function does the same as standard atof(), but it allows a variety of locale representations. That is it supports numeric values with either a comma or a period for the decimal delimiter.

PS. The M stands for Multi-lingual.

Parameters:

nptr The string to convert.

Returns:

Converted value, if any. Zero on failure.

References CPLAtofM(), and CPLStrtodDelim().

Referenced by CPLAtofM(), GDALLoadWorldFile(), GDALRasterBand::GetMaximum(), and GDALRasterBand::GetMinimum().

44.1.2.4 void* CPLCalloc (size_t *nCount*, size_t *nSize*)

Safe version of calloc().

This function is like the C library calloc(), but raises a CE_Fatal error with CPLError() if it fails to allocate the desired memory. It should be used for small memory allocations that are unlikely to fail and for which the application is unwilling to test for out of memory conditions. It uses VSICalloc() to get the memory, so any hooking of VSICalloc() will apply to **CPLCalloc()** (p. **??**) as well. CPLFree() or VSIFree() can be used free memory allocated by **CPLCalloc()** (p. **??**).

Parameters:

nCount number of objects to allocate.

nSize size (in bytes) of object to allocate.

Returns:

pointer to newly allocated memory, only NULL if nSize * nCount is NULL.

44.1.2.5 int CPLCheckForFile (char * *pszFilename*, char ** *papszSiblingFiles*)

Check for file existance.

The function checks if a named file exists in the filesystem, hopefully in an efficient fashion if a sibling file list is available. It exists primarily to do faster file checking for functions like GDAL open methods that get a list of files from the target directory.

If the sibling file list exists (is not NULL) it is assumed to be a list of files in the same directory as the target file, and it will be checked (case insensitively) for a match. If a match is found, pszFilename is updated with the correct case and TRUE is returned.

If papszSiblingFiles is NULL, a **VSIStatL()** (p. **??**) is used to test for the files existance, and no case insensitive testing is done.

Parameters:

pszFilename name of file to check for - filename case updated in some cases.

papszSiblingFiles a list of files in the same directory as pszFilename if available, or NULL. This list should have no path components.

Returns:

TRUE if a match is found, or FALSE if not.

References CPLGetFilename(), and VSIStatL().

44.1.2.6 const char* CPLCleanTrailingSlash (const char * *pszPath*)

Remove trailing forward/backward slash from the path for unix/windows resp.

Returns a string containing the portion of the passed path string with trailing slash removed. If there is no path in the passed filename an empty string will be returned (not NULL).

```
CPLCleanTrailingSlash( "abc/def/" ) == "abc/def"
CPLCleanTrailingSlash( "abc/def" ) == "abc/def"
CPLCleanTrailingSlash( "c:\abc\def\" ) == "c:\abc\def"
CPLCleanTrailingSlash( "c:\abc\def" ) == "c:\abc\def"
CPLCleanTrailingSlash( "abc" ) == "abc"
```

Parameters:

pszPath the path to be cleaned up

Returns:

Path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call. The returned will generally not contain a trailing path separator.

References CPLCleanTrailingSlash().

Referenced by CPLCleanTrailingSlash().

44.1.2.7 void CPLCloseShared (FILE * *fp*)

Close shared file.

Dereferences the indicated file handle, and closes it if the reference count has dropped to zero. A CPLError() is issued if the file is not in the shared file list.

Parameters:

fp file handle from **CPLOpenShared()** (p. **??**) to deaccess.

References VSIFCloseL().

44.1.2.8 char** CPLCorrespondingPaths (const char * *pszOldFilename*, const char * *pszNewFilename*, char ** *papszFileList*)

Identify corresponding paths.

Given a prototype old and new filename this function will attempt to determine corresponding names for a set of other old filenames that will rename them in a similar manner. This correspondance assumes there are two possibly kinds of renaming going on. A change of path, and a change of filename stem.

If a consistent renaming cannot be established for all the files this function will return indicating an error.

The returned file list becomes owned by the caller and should be destroyed with **CSLDestroy()** (p. **??**).

Parameters:

pszOldFilename path to old prototype file.

pszNewFilename path to new prototype file.

papszFileList list of other files associated with pszOldFilename to rename similarly.

Returns:

a list of files corresponding to papszFileList but renamed to correspond to pszNewFilename.

References CPLCorrespondingPaths(), CPLFormFilename(), CPLGetBasename(), CPLGetFilename(), and CPLGetPath().

Referenced by GDALDriver::CopyFiles(), CPLCorrespondingPaths(), and GDALDriver::Rename().

44.1.2.9 double CPLDecToPackedDMS (double *dfDec*)

Convert decimal degrees into packed DMS value (DDDMMMSSS.SS).

This function converts a value, specified in decimal degrees into packed DMS angle. The standard packed DMS format is:

degrees * 1000000 + minutes * 1000 + seconds

See also **CPLPackedDMSToDec()** (p. **??**).

Parameters:

dfDec Angle in decimal degrees.

Returns:

Angle in packed DMS format.

44.1.2.10 void CPLDumpSharedList (FILE * *fp*)

Report open shared files.

Dumps all open shared files to the indicated file handle. If the file handle is NULL information is sent via the CPLDebug() call.

Parameters:

fp File handle to write to.

44.1.2.11 const char* CPLExtractRelativePath (const char * *pszBaseDir*, const char * *pszTarget*, int * *pbGotRelative*)

Get relative path from directory to target file.

Computes a relative path for pszTarget relative to pszBaseDir. Currently this only works if they share a common base path. The returned path is normally into the pszTarget string. It should only be considered valid as long as pszTarget is valid or till the next call to this function, whichever comes first.

Parameters:

pszBaseDir the name of the directory relative to which the path should be computed. pszBaseDir may be NULL in which case the original target is returned without relitivizing.

pszTarget the filename to be changed to be relative to pszBaseDir.

pbGotRelative Pointer to location in which a flag is placed indicating that the returned path is relative to the basename (TRUE) or not (FALSE). This pointer may be NULL if flag is not desired.

Returns:

an adjusted path or the original if it could not be made relative to the pszBaseFile's path.

References CPLExtractRelativePath(), and CPLGetPath().

Referenced by CPLExtractRelativePath().

44.1.2.12 char* CPLFGets (char * *pszBuffer*, int *nBufferSize*, FILE * *fp*)

Reads in at most one less than nBufferSize characters from the fp stream and stores them into the buffer pointed to by pszBuffer. Reading stops after an EOF or a newline. If a newline is read, it is _not_ stored into the buffer. A '\0' is stored after the last character in the buffer. All three types of newline terminators recognized by the **CPLFGets()** (p. **??**): single '\r' and '\n' and '\r\n' combination.

Parameters:

pszBuffer pointer to the targeting character buffer.

nBufferSize maximum size of the string to read (not including termonating '\0').

fp file pointer to read from.

Returns:

pointer to the pszBuffer containing a string read from the file or NULL if the error or end of file was encountered.

44.1.2.13 const char* CPLFormCIFilename (const char * *pszPath*, const char * *pszBasename*, const char * *pszExtension*)

Case insensitive file searching, returing full path.

This function tries to return the path to a file regardless of whether the file exactly matches the basename, and extension case, or is all upper case, or all lower case. The path is treated as case sensitive. This function is equivelent to **CPLFormFilename()** (p. **??**) on case insensitive file systems (like Windows).

Parameters:

pszPath directory path to the directory containing the file. This may be relative or absolute, and may have a trailing path separator or not. May be NULL.

pszBasename file basename. May optionally have path and/or extension. May not be NULL.

pszExtension file extension, optionally including the period. May be NULL.

Returns:

a fully formed filename in an internal static string. Do not modify or free the returned string. The string may be destroyed by the next CPL call.

References CPLFormCIFilename(), CPLFormFilename(), and VSIStatL().

Referenced by CPLFormCIFilename().

44.1.2.14 const char* CPLFormFilename (const char * *pszPath*, const char * *pszBasename*, const char * *pszExtension*)

Build a full file path from a passed path, file basename and extension.

The path, and extension are optional. The basename may in fact contain an extension if desired.

```
CPLFormFilename("abc/xyz","def", ".dat" ) == "abc/xyz/def.dat"
CPLFormFilename(NULL,"def", NULL ) == "def"
CPLFormFilename(NULL,"abc/def.dat", NULL ) == "abc/def.dat"
CPLFormFilename("/abc/xyz/","def.dat", NULL ) == "/abc/xyz/def.dat"
```

Parameters:

pszPath directory path to the directory containing the file. This may be relative or absolute, and may have a trailing path separator or not. May be NULL.

pszBasename file basename. May optionally have path and/or extension. May not be NULL.

pszExtension file extension, optionally including the period. May be NULL.

Returns:

a fully formed filename in an internal static string. Do not modify or free the returned string. The string may be destroyed by the next CPL call.

References CPLFormFilename().

Referenced by GDALDriverManager::AutoLoadDrivers(), CPLCorrespondingPaths(), CPLFormCIFilename(), CPLFormFilename(), CPLGenerateTempFilename(), CPLUnlinkTree(), and GDALGeneralCmdLineProcessor().

44.1.2.15 const char* CPLGenerateTempFilename (const char * *pszStem*)

Generate temporary file name.

Returns a filename that may be used for a temporary file. The location of the file tries to follow operating system semantics but may be forced via the CPL_TMPDIR configuration option.

Parameters:

pszStem if non-NULL this will be part of the filename.

Returns:

a filename which is valid till the next CPL call in this thread.

References CPLFormFilename(), and CPLGenerateTempFilename().

Referenced by CPLGenerateTempFilename().

44.1.2.16 const char* CPLGetBasename (const char * *pszFullFilename*)

Extract basename (non-directory, non-extension) portion of filename.

Returns a string containing the file basename portion of the passed name. If there is no basename (passed value ends in trailing directory separator, or filename starts with a dot) an empty string is returned.

```
CPLGetBasename( "abc/def.xyz" ) == "def"
CPLGetBasename( "abc/def" ) == "def"
CPLGetBasename( "abc/def/" ) == ""
```

Parameters:

pszFullFilename the full filename potentially including a path.

Returns:

just the non-directory, non-extension portion of the path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call.

References CPLGetBasename().

Referenced by GDALDriverManager::AutoLoadDrivers(), CPLCorrespondingPaths(), and CPLGetBasename().

44.1.2.17 char* CPLGetCurrentDir (void)

Get the current working directory name.

Returns:

a pointer to buffer, containing current working directory path or NULL in case of error. User is responsible to free that buffer after usage with CPLFree() function. If HAVE_GETCWD macro is not defined, the function returns NULL.

References CPLGetCurrentDir().

Referenced by CPLGetCurrentDir().

44.1.2.18 const char* CPLGetDirname (const char * *pszFilename*)

Extract directory path portion of filename.

Returns a string containing the directory path portion of the passed filename. If there is no path in the passed filename the dot will be returned. It is the only difference from **CPLGetPath()** (p. **??**).

```
CPLGetDirname( "abc/def.xyz" ) == "abc"
CPLGetDirname( "/abc/def/" ) == "/abc/def"
CPLGetDirname( "/" ) == "/"
CPLGetDirname( "/abc/def" ) == "/abc"
CPLGetDirname( "abc" ) == "."
```

Parameters:

pszFilename the filename potentially including a path.

Returns:

Path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call. The returned will generally not contain a trailing path separator.

References CPLGetDirname().

Referenced by GDALDriverManager::AutoLoadDrivers(), and CPLGetDirname().

44.1.2.19 int CPLGetExecPath (char * *pszPathBuf*, int *nMaxLength*)

Fetch path of executable.

The path to the executable currently running is returned. This path includes the name of the executable. Currently this only works on win32 platform.

Parameters:

pszPathBuf the buffer into which the path is placed.

nMaxLength the buffer size, MAX_PATH+1 is suggested.

Returns:

FALSE on failure or TRUE on success.

References CPLGetExecPath().

Referenced by GDALDriverManager::AutoLoadDrivers(), and CPLGetExecPath().

44.1.2.20 const char* CPLGetExtension (const char * *pszFullFilename*)

Extract filename extension from full filename.

Returns a string containing the extention portion of the passed name. If there is no extension (the filename has no dot) an empty string is returned. The returned extension will not include the period.

```
CPLGetExtension( "abc/def.xyz" ) == "xyz"
CPLGetExtension( "abc/def" ) == ""
```

Parameters:

pszFullFilename the full filename potentially including a path.

Returns:

just the extension portion of the path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call.

References CPLGetExtension().

Referenced by GDALDriverManager::AutoLoadDrivers(), CPLGetExtension(), and GDALReadWorldFile().

44.1.2.21 const char* CPLGetFilename (const char * *pszFullFilename*)

Extract non-directory portion of filename.

Returns a string containing the bare filename portion of the passed filename. If there is no filename (passed value ends in trailing directory separator) an empty string is returned.

```
CPLGetFilename( "abc/def.xyz" ) == "def.xyz"
CPLGetFilename( "/abc/def/" ) == ""
CPLGetFilename( "abc/def" ) == "def"
```

Parameters:

pszFullFilename the full filename potentially including a path.

Returns:

just the non-directory portion of the path (points back into original string).

References CPLGetFilename().

Referenced by CPLCheckForFile(), CPLCorrespondingPaths(), and CPLGetFilename().

44.1.2.22 const char* CPLGetPath (const char * *pszFilename*)

Extract directory path portion of filename.

Returns a string containing the directory path portion of the passed filename. If there is no path in the passed filename an empty string will be returned (not NULL).

```
CPLGetPath( "abc/def.xyz" ) == "abc"
CPLGetPath( "/abc/def/" ) == "/abc/def"
CPLGetPath( "/" ) == "/"
CPLGetPath( "/abc/def" ) == "/abc"
CPLGetPath( "abc" ) == ""
```

Parameters:

pszFilename the filename potentially including a path.

Returns:

Path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call. The returned will generally not contain a trailing path separator.

References CPLGetPath().

Referenced by CPLCorrespondingPaths(), CPLExtractRelativePath(), and CPLGetPath().

44.1.2.23 CPLSharedFileInfo* CPLGetSharedList (int * *pnCount*)

Fetch list of open shared files.

Parameters:

pnCount place to put the count of entries.

Returns:

the pointer to the first in the array of shared file info structures.

44.1.2.24 void* CPLGetSymbol (const char * *pszLibrary*, const char * *pszSymbolName*)

Fetch a function pointer from a shared library / DLL.

This function is meant to abstract access to shared libraries and DLLs and performs functions similar to dlopen()/dlsym() on Unix and LoadLibrary() / GetProcAddress() on Windows.

If no support for loading entry points from a shared library is available this function will always return NULL. Rules on when this function issues a CPLError() or not are not currently well defined, and will have to be resolved in the future.

Currently **CPLGetSymbol()** (p. **??**) doesn't try to:

- prevent the reference count on the library from going up for every request, or given any opportunity to unload the library.
- Attempt to look for the library in non-standard locations.
- Attempt to try variations on the symbol name, like pre-prending or post-pending an underscore.

Some of these issues may be worked on in the future.

Parameters:

pszLibrary the name of the shared library or DLL containing the function. May contain path to file. If not system supplies search paths will be used.

pszSymbolName the name of the function to fetch a pointer to.

Returns:

A pointer to the function if found, or NULL if the function isn't found, or the shared library can't be loaded.

References CPLGetSymbol().

Referenced by GDALDriverManager::AutoLoadDrivers(), and CPLGetSymbol().

44.1.2.25 int CPLIsFilenameRelative (const char * *pszFilename*)

Is filename relative or absolute?

The test is filesystem convention agnostic. That is it will test for Unix style and windows style path conventions regardless of the actual system in use.

Parameters:

pszFilename the filename with path to test.

Returns:

TRUE if the filename is relative or FALSE if it is absolute.

References CPLIsFilenameRelative().

Referenced by CPLIsFilenameRelative(), and CPLProjectRelativeFilename().

44.1.2.26 void* CPLMalloc (size_t *nSize*)

Safe version of malloc().

This function is like the C library malloc(), but raises a CE_Fatal error with CPLError() if it fails to allocate the desired memory. It should be used for small memory allocations that are unlikely to fail and for which the application is unwilling to test for out of memory conditions. It uses VSIMalloc() to get the memory, so any hooking of VSIMalloc() will apply to **CPLMalloc()** (p. **??**) as well. CPLFree() or VSIFree() can be used free memory allocated by **CPLMalloc()** (p. **??**).

Parameters:

nSize size (in bytes) of memory block to allocate.

Returns:

pointer to newly allocated memory, only NULL if nSize is zero.

44.1.2.27 FILE* CPLOpenShared (const char * *pszFilename*, const char * *pszAccess*, int *bLarge*)

Open a shared file handle.

Some operating systems have limits on the number of file handles that can be open at one time. This function attempts to maintain a registry of already open file handles, and reuse existing ones if the same file is requested by another part of the application.

Note that access is only shared for access types "r", "rb", "r+" and "rb+". All others will just result in direct VSIOpen() calls. Keep in mind that a file is only reused if the file name is exactly the same. Different names referring to the same file will result in different handles.

The VSIFOpen() or **VSIFOpenL()** (p. **??**) function is used to actually open the file, when an existing file handle can't be shared.

Parameters:

pszFilename the name of the file to open.

pszAccess the normal fopen()/VSIFOpen() style access string.

bLarge If TRUE **VSIFOpenL()** (p. **??**) (for large files) will be used instead of VSIFOpen().

Returns:

a file handle or NULL if opening fails.

References VSIFOpenL().

44.1.2.28 double CPLPackedDMSToDec (double *dfPacked*)

Convert a packed DMS value (DDDMMMSSS.SS) into decimal degrees.

This function converts a packed DMS angle to seconds. The standard packed DMS format is:

degrees $*$ 1000000 + minutes $*$ 1000 + seconds

Example: ang = 120025045.25 yields deg = 120 min = 25 sec = 45.25

The algorithm used for the conversion is as follows:

1. The absolute value of the angle is used.

2. The degrees are separated out: deg = ang/1000000 (fractional portion truncated)

3. The minutes are separated out: min = (ang - deg $*$ 1000000) / 1000 (fractional portion truncated)

4. The seconds are then computed: sec = ang - deg $*$ 1000000 - min $*$ 1000

5. The total angle in seconds is computed: sec = deg $*$ 3600.0 + min $*$ 60.0 + sec

6. The sign of sec is set to that of the input angle.

Packed DMS values used by the USGS GCTP package and probably by other software.

NOTE: This code does not validate input value. If you give the wrong value, you will get the wrong result.

Parameters:

dfPacked Angle in packed DMS format.

Returns:

Angle in decimal degrees.

44.1.2.29 int CPLPrintDouble (char $*$ *pszBuffer*, const char $*$ *pszFormat*, double *dfValue*, const char $*$ *pszLocale*)

Print double value into specified string buffer. Exponential character flag 'E' (or 'e') will be replaced with 'D', as in Fortran. Resulting string will not to be NULL-terminated.

Parameters:

pszBuffer Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.

pszFormat Format specifier (for example, "%16.9E").

dfValue Numerical value to print.

pszLocale Pointer to a character string containing locale name ("C", "POSIX", "us_US", "ru_RU.KOI8-R" etc.). If NULL we will not manipulate with locale settings and current process locale will be used for printing. With the pszLocale option we can control what exact locale will be used for printing a numeric value to the string (in most cases it should be C/POSIX).

Returns:

Number of characters printed.

44.1.2.30 int CPLPrintInt32 (char * *pszBuffer*, GInt32 *iValue*, int *nMaxLen*)

Print GInt32 value into specified string buffer. This string will not be NULL-terminated.

Parameters:

pszBuffer Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.

iValue Numerical value to print.

nMaxLen Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns:

Number of characters printed.

44.1.2.31 int CPLPrintPointer (char * *pszBuffer*, void * *pValue*, int *nMaxLen*)

Print pointer value into specified string buffer. This string will not be NULL-terminated.

Parameters:

pszBuffer Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.

pValue Pointer to ASCII encode.

nMaxLen Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns:

Number of characters printed.

44.1.2.32 int CPLPrintString (char * *pszDest*, const char * *pszSrc*, int *nMaxLen*)

Copy the string pointed to by pszSrc, NOT including the terminating '\0' character, to the array pointed to by pszDest.

Parameters:

pszDest Pointer to the destination string buffer. Should be large enough to hold the resulting string.

pszSrc Pointer to the source buffer.

nMaxLen Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns:

Number of characters printed.

44.1.2.33 int CPLPrintStringFill (char * *pszDest*, const char * *pszSrc*, int *nMaxLen*)

Copy the string pointed to by pszSrc, NOT including the terminating '\0' character, to the array pointed to by pszDest. Remainder of the destination string will be filled with space characters. This is only difference from the PrintString().

Parameters:

pszDest Pointer to the destination string buffer. Should be large enough to hold the resulting string.

pszSrc Pointer to the source buffer.

nMaxLen Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns:

Number of characters printed.

44.1.2.34 int CPLPrintTime (char * *pszBuffer*, int *nMaxLen*, const char * *pszFormat*, const struct tm * *poBrokenTime*, const char * *pszLocale*)

Print specified time value accordingly to the format options and specified locale name. This function does following:

- if locale parameter is not NULL, the current locale setting will be stored and replaced with the specified one;
- format time value with the strftime(3) function;
- restore back current locale, if was saved.

Parameters:

pszBuffer Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.

nMaxLen Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

pszFormat Controls the output format. Options are the same as for strftime(3) function.

poBrokenTime Pointer to the broken-down time structure. May be requested with the VSIGMTime() and VSILocalTime() functions.

pszLocale Pointer to a character string containing locale name ("C", "POSIX", "us_US", "ru_RU.KOI8-R" etc.). If NULL we will not manipulate with locale settings and current process locale will be used for printing. Be aware that it may be unsuitable to use current locale for printing time, because all names will be printed in your native language, as well as time format settings also may be ajusted differently from the C/POSIX defaults. To solve these problems this option was introdiced.

Returns:

Number of characters printed.

44.1.2.35 int CPLPrintUIntBig (char * *pszBuffer*, GUIntBig *iValue*, int *nMaxLen*)

Print GUIntBig value into specified string buffer. This string will not be NULL-terminated.

Parameters:

pszBuffer Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.

iValue Numerical value to print.

nMaxLen Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

Returns:

Number of characters printed.

44.1.2.36 const char* CPLProjectRelativeFilename (const char * *pszProjectDir*, const char * *pszSecondaryFilename*)

Find a file relative to a project file.

Given the path to a "project" directory, and a path to a secondary file referenced from that project, build a path to the secondary file that the current application can use. If the secondary path is already absolute, rather than relative, then it will be returned unaltered.

Examples:

```
CPLProjectRelativeFilename("abc/def","tmp/abc.gif") == "abc/def/tmp/abc.gif"
CPLProjectRelativeFilename("abc/def","/tmp/abc.gif") == "/tmp/abc.gif"
CPLProjectRelativeFilename("/xy", "abc.gif") == "/xy/abc.gif"
CPLProjectRelativeFilename("/abc/def","../abc.gif") == "/abc/def/../abc.gif"
CPLProjectRelativeFilename("C:\WIN","abc.gif") == "C:\WIN\abc.gif"
```

Parameters:

pszProjectDir the directory relative to which the secondary files path should be interpreted.

pszSecondaryFilename the filename (potentially with path) that is to be interpreted relative to the project directory.

Returns:

a composed path to the secondary file. The returned string is internal and should not be altered, freed, or depending on past the next CPL call.

References CPLIsFilenameRelative(), and CPLProjectRelativeFilename().

Referenced by CPLProjectRelativeFilename().

44.1.2.37 const char* CPLReadLine (FILE * *fp*)

Simplified line reading from text file.

Read a line of text from the given file handle, taking care to capture CR and/or LF and strip off ... equivelent of DKReadLine(). Pointer to an internal buffer is returned. The application shouldn't free it, or depend on it's value past the next call to **CPLReadLine()** (p. **??**).

Note that **CPLReadLine()** (p. **??**) uses VSIFGets(), so any hooking of VSI file services should apply to **CPLReadLine()** (p. **??**) as well.

CPLReadLine() (p. **??**) maintains an internal buffer, which will appear as a single block memory leak in some circumstances. **CPLReadLine()** (p. **??**) may be called with a NULL FILE * at any time to free this working buffer.

Parameters:

fp file pointer opened with VSIFOpen().

Returns:

pointer to an internal buffer containing a line of text read from the file or NULL if the end of file was encountered.

44.1.2.38 const char* CPLReadLineL (FILE * *fp*)

Simplified line reading from text file.

Similar to **CPLReadLine()** (p. **??**), but reading from a large file API handle.

Parameters:

fp file pointer opened with **VSIFOpenL()** (p. **??**).

Returns:

pointer to an internal buffer containing a line of text read from the file or NULL if the end of file was encountered.

References VSIFReadL(), VSIFSeekL(), and VSIFTellL().

44.1.2.39 void* CPLRealloc (void * *pData*, size_t *nNewSize*)

Safe version of realloc().

This function is like the C library realloc(), but raises a CE_Fatal error with CPLError() if it fails to allocate the desired memory. It should be used for small memory allocations that are unlikely to fail and for which

the application is unwilling to test for out of memory conditions. It uses VSIRealloc() to get the memory, so any hooking of VSIRealloc() will apply to **CPLRealloc()** (p. **??**) as well. CPLFree() or VSIFree() can be used free memory allocated by **CPLRealloc()** (p. **??**).

It is also safe to pass NULL in as the existing memory block for **CPLRealloc()** (p. **??**), in which case it uses VSIMalloc() to allocate a new block.

Parameters:

pData existing memory block which should be copied to the new block.

nNewSize new size (in bytes) of memory block to allocate.

Returns:

pointer to allocated memory, only NULL if nNewSize is zero.

44.1.2.40 const char* CPLResetExtension (const char * *pszPath*, const char * *pszExt*)

Replace the extension with the provided one.

Parameters:

pszPath the input path, this string is not altered.

pszExt the new extension to apply to the given path.

Returns:

an altered filename with the new extension. Do not modify or free the returned string. The string may be destroyed by the next CPL call.

References CPLResetExtension().

Referenced by CPLResetExtension(), GDALReadWorldFile(), and GDALWriteWorldFile().

44.1.2.41 double CPLScanDouble (const char * *pszString*, int *nMaxLength*)

Extract double from string.

Scan up to a maximum number of characters from a string and convert the result to a double. This function uses **CPLAtof()** (p. **??**) to convert string to double value, so it uses a comma as a decimal delimiter.

Parameters:

pszString String containing characters to be scanned. It may be terminated with a null character.

nMaxLength The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns:

Double value, converted from its ASCII form.

References CPLAtof().

44.1.2.42 long CPLScanLong (const char * *pszString*, int *nMaxLength*)

Scan up to a maximum number of characters from a string and convert the result to a long.

Parameters:

pszString String containing characters to be scanned. It may be terminated with a null character.

nMaxLength The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns:

Long value, converted from its ASCII form.

44.1.2.43 void* CPLScanPointer (const char * *pszString*, int *nMaxLength*)

Extract pointer from string.

Scan up to a maximum number of characters from a string and convert the result to a pointer.

Parameters:

pszString String containing characters to be scanned. It may be terminated with a null character.

nMaxLength The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns:

pointer value, converted from its ASCII form.

44.1.2.44 char* CPLScanString (const char * *pszString*, int *nMaxLength*, int *bTrimSpaces*, int *bNormalize*)

Scan up to a maximum number of characters from a given string, allocate a buffer for a new string and fill it with scanned characters.

Parameters:

pszString String containing characters to be scanned. It may be terminated with a null character.

nMaxLength The maximum number of character to read. Less characters will be read if a null character is encountered.

bTrimSpaces If TRUE, trim ending spaces from the input string. Character considered as empty using isspace(3) function.

bNormalize If TRUE, replace ':' symbol with the '_'. It is needed if resulting string will be used in CPL dictionaries.

Returns:

Pointer to the resulting string buffer. Caller responsible to free this buffer with CPLFree().

44.1.2.45 GUIntBig CPLScanUIntBig (const char * *pszString*, int *nMaxLength*)

Extract big integer from string.

Scan up to a maximum number of characters from a string and convert the result to a GUIntBig.

Parameters:

pszString String containing characters to be scanned. It may be terminated with a null character.

nMaxLength The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns:

GUIntBig value, converted from its ASCII form.

44.1.2.46 unsigned long CPLScanULong (const char * *pszString*, int *nMaxLength*)

Scan up to a maximum number of characters from a string and convert the result to a unsigned long.

Parameters:

pszString String containing characters to be scanned. It may be terminated with a null character.

nMaxLength The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

Returns:

Unsigned long value, converted from its ASCII form.

44.1.2.47 char* CPLStrdup (const char * *pszString*)

Safe version of strdup() function.

This function is similar to the C library strdup() function, but if the memory allocation fails it will issue a CE_Fatal error with CPLError() instead of returning NULL. It uses VSIStrdup(), so any hooking of that function will apply to **CPLStrdup()** (p. **??**) as well. Memory allocated with **CPLStrdup()** (p. **??**) can be freed with CPLFree() or VSIFree().

It is also safe to pass a NULL string into **CPLStrdup()** (p. **??**). **CPLStrdup()** (p. **??**) will allocate and return a zero length string (as opposed to a NULL string).

Parameters:

pszString input string to be duplicated. May be NULL.

Returns:

pointer to a newly allocated copy of the string. Free with CPLFree() or VSIFree().

44.1.2.48 char* CPLStrlwr (char * *pszString*)

Convert each characters of the string to lower case.

For example, "ABcdE" will be converted to "abcde". This function is locale dependent.

Parameters:

pszString input string to be converted.

Returns:

pointer to the same string, pszString.

44.1.2.49 double CPLStrtod (const char * *nptr*, char ** *endptr*)

Converts ASCII string to floating point number.

This function converts the initial portion of the string pointed to by nptr to double floating point representation. This function does the same as standard strtod(3), but does not take locale in account. That means, the decimal delimiter is always '.' (decimal point). Use **CPLStrtodDelim()** (p. **??**) function if you want to specify custom delimiter. Also see notes for **CPLAtof()** (p. **??**) function.

Parameters:

nptr Pointer to string to convert.

endptr If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by endptr.

Returns:

Converted value, if any.

References CPLStrtod(), and CPLStrtodDelim().

Referenced by CPLAtof(), CPLStrtod(), and GDALValidateCreationOptions().

44.1.2.50 double CPLStrtodDelim (const char * *nptr*, char ** *endptr*, char *point*)

Converts ASCII string to floating point number using specified delimiter.

This function converts the initial portion of the string pointed to by nptr to double floating point representation. This function does the same as standard strtod(3), but does not take locale in account. Instead of locale defined decimal delimiter you can specify your own one. Also see notes for **CPLAtof()** (p. **??**) function.

Parameters:

nptr Pointer to string to convert.

endptr If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by endptr.

point Decimal delimiter.

Returns:

Converted value, if any.

References CPLStrtodDelim().

Referenced by CPLAtofDelim(), CPLAtofM(), CPLStrtod(), CPLStrtodDelim(), and CPLStrtofDelim().

44.1.2.51 float CPLStrtof (const char * *nptr*, char ** *endptr*)

Converts ASCII string to floating point number.

This function converts the initial portion of the string pointed to by nptr to single floating point representation. This function does the same as standard strtof(3), but does not take locale in account. That means, the decimal delimiter is always '.' (decimal point). Use **CPLStrtofDelim()** (p. **??**) function if you want to specify custom delimiter. Also see notes for **CPLAtof()** (p. **??**) function.

Parameters:

nptr Pointer to string to convert.

endptr If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by endptr.

Returns:

Converted value, if any.

References CPLStrtof(), and CPLStrtofDelim().

Referenced by CPLStrtof().

44.1.2.52 float CPLStrtofDelim (const char * *nptr*, char ** *endptr*, char *point*)

Converts ASCII string to floating point number using specified delimiter.

This function converts the initial portion of the string pointed to by nptr to single floating point representation. This function does the same as standard strtof(3), but does not take locale in account. Instead of locale defined decimal delimiter you can specify your own one. Also see notes for **CPLAtof()** (p. **??**) function.

Parameters:

nptr Pointer to string to convert.

endptr If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by endptr.

point Decimal delimiter.

Returns:

Converted value, if any.

References CPLStrtodDelim(), and CPLStrtofDelim().

Referenced by CPLStrtof(), and CPLStrtofDelim().

44.1.2.53 int CPLUnlinkTree (const char * *pszPath*)

Returns:

0 on successful completion, -1 if function fails.

References CPLFormFilename(), VSIRmdir(), and VSIUnlink().

44.2 cpl_error.h File Reference

```
#include "cpl_port.h"
```

Defines

- #define **CPLAssert**(expr)
- #define **VALIDATE_POINTER_ERR** CE_Failure
- #define **VALIDATE_POINTER0**(ptr, func)
- #define **VALIDATE_POINTER1**(ptr, func, rc)
- #define **CPLE_None** 0
- #define **CPLE_AppDefined** 1
- #define **CPLE_OutOfMemory** 2
- #define **CPLE_FileIO** 3
- #define **CPLE_OpenFailed** 4
- #define **CPLE_IllegalArg** 5
- #define **CPLE_NotSupported** 6
- #define **CPLE_AssertionFailed** 7
- #define **CPLE_NoWriteAccess** 8
- #define **CPLE_UserInterrupt** 9
- #define **CPLE_ObjectNull** 10

Typedefs

- typedef const char ∗void **CPLLoggingErrorHandler** (CPLErr, int, const char ∗)

Enumerations

- enum **CPLErr** {
  CE_None = 0, **CE_Debug** = 1, **CE_Warning** = 2, **CE_Failure** = 3,
  CE_Fatal = 4 }

Functions

- void **CPLError** (CPLErr eErrClass, int err_no, const char ∗fmt,...) CPL_PRINT_FUNC_-FORMAT(3
- void void **CPLErrorV** (CPLErr, int, const char ∗, va_list)
- void **CPLErrorReset** (void)
- int **CPLGetLastErrorNo** (void)
- CPLErr **CPLGetLastErrorType** (void)
- const char ∗ **CPLGetLastErrorMsg** (void)
- typedef **void** (1 ∗CPLErrorHandler)(CPLErr
- void **CPLDefaultErrorHandler** (CPLErr, int, const char ∗)
- void **CPLQuietErrorHandler** (CPLErr, int, const char ∗)
- CPLErrorHandler **CPLSetErrorHandler** (CPLErrorHandler)
- void **CPLPushErrorHandler** (CPLErrorHandler)
- void **CPLPopErrorHandler** (void)
- void **CPLDebug** (const char ∗, const char ∗,...) CPL_PRINT_FUNC_FORMAT(2
- void void **_CPLAssert** (const char ∗, const char ∗, int)

Variables

- typedef **int**

44.2.1 Detailed Description

CPL error handling services.

44.2.2 Define Documentation

44.2.2.1 #define VALIDATE_POINTER0(ptr, func)

Value:

```
do { if( NULL == ptr ) \
      { \
        CPLErr const ret = VALIDATE_POINTER_ERR; \
        CPLError( ret, CPLE_ObjectNull, \
           "Pointer \'%s\' is NULL in \'%s\'.\n", #ptr, (func)); \
         return; }} while(0)
```

44.2.2.2 #define VALIDATE_POINTER1(ptr, func, rc)

Value:

```
do { if( NULL == ptr ) \
      { \
          CPLErr const ret = VALIDATE_POINTER_ERR; \
          CPLError( ret, CPLE_ObjectNull, \
            "Pointer \'%s\' is NULL in \'%s\'.\n", #ptr, (func)); \
        return (rc); }} while(0)
```

44.2.3 Function Documentation

44.2.3.1 void void _CPLAssert (const char * *pszExpression*, const char * *pszFile*, int *iLine*)

Report failure of a logical assertion.

Applications would normally use the CPLAssert() macro which expands into code calling **_CPLAssert()** (p. **??**) only if the condition fails. **_CPLAssert()** (p. **??**) will generate a CE_Fatal error call to CPLError(), indicating the file name, and line number of the failed assertion, as well as containing the assertion itself.

There is no reason for application code to call **_CPLAssert()** (p. **??**) directly.

44.2.3.2 void CPLErrorReset (void)

Erase any traces of previous errors.

This is normally used to ensure that an error which has been recovered from does not appear to be still in play with high level functions.

44.2.3.3 const char* CPLGetLastErrorMsg (void)

Get the last error message.

Fetches the last error message posted with CPLError(), that hasn't been cleared by **CPLErrorReset()** (p. **??**). The returned pointer is to an internal string that should not be altered or freed.

Returns:

the last error message, or NULL if there is no posted error message.

44.2.3.4 int CPLGetLastErrorNo (void)

Fetch the last error number.

This is the error number, not the error class.

Returns:

the error number of the last error to occur, or CPLE_None (0) if there are no posted errors.

44.2.3.5 CPLErr CPLGetLastErrorType (void)

Fetch the last error type.

This is the error class, not the error number.

Returns:

the error number of the last error to occur, or CE_None (0) if there are no posted errors.

44.2.3.6 void CPLPopErrorHandler (void)

Pop error handler off stack.

Discards the current error handler on the error handler stack, and restores the one in use before the last **CPLPushErrorHandler()** (p. **??**) call. This method has no effect if there are no error handlers on the current threads error handler stack.

44.2.3.7 void CPLPushErrorHandler (CPLErrorHandler *pfnErrorHandlerNew*)

Push a new CPLError handler.

This pushes a new error handler on the thread-local error handler stack. This handler will be used untill removed with **CPLPopErrorHandler()** (p. **??**).

The **CPLSetErrorHandler()** (p. **??**) docs have further information on how CPLError handlers work.

Parameters:

pfnErrorHandlerNew new error handler function.

44.2.3.8 CPLErrorHandler CPLSetErrorHandler (CPLErrorHandler *pfnErrorHandlerNew*)

Install custom error handler.

Allow the library's user to specify his own error handler function. A valid error handler is a C function with the following prototype:

```
void MyErrorHandler(CPLErr eErrClass, int err_no, const char *msg)
```

Pass NULL to come back to the default behavior. The default behaviour (CPLDefaultErrorHandler()) is to write the message to stderr.

The msg will be a partially formatted error message not containing the "ERROR %d:" portion emitted by the default handler. Message formatting is handled by CPLError() before calling the handler. If the error handler function is passed a CE_Fatal class error and returns, then CPLError() will call abort(). Applications wanting to interrupt this fatal behaviour will have to use longjmp(), or a C++ exception to indirectly exit the function.

Another standard error handler is CPLQuietErrorHandler() which doesn't make any attempt to report the passed error or warning messages but will process debug messages via CPLDefaultErrorHandler.

Note that error handlers set with **CPLSetErrorHandler()** (p. **??**) apply to all threads in an application, while error handlers set with CPLPushErrorHandler are thread-local. However, any error handlers pushed with CPLPushErrorHandler (and not removed with CPLPopErrorHandler) take precidence over the global error handlers set with **CPLSetErrorHandler()** (p. **??**). Generally speaking **CPLSetErrorHandler()** (p. **??**) would be used to set a desired global error handler, while **CPLPushErrorHandler()** (p. **??**) would be used to install a temporary local error handler, such as CPLQuietErrorHandler() to suppress error reporting in a limited segment of code.

Parameters:

pfnErrorHandlerNew new error handler function.

Returns:

returns the previously installed error handler.

44.3 cpl_hash_set.h File Reference

```
#include "cpl_port.h"
```

Typedefs

- typedef struct _CPLHashSet **CPLHashSet**
- typedef unsigned long(∗ **CPLHashSetHashFunc**)(const void ∗elt)
- typedef int(∗ **CPLHashSetEqualFunc**)(const void ∗elt1, const void ∗elt2)
- typedef void(∗ **CPLHashSetFreeEltFunc**)(void ∗elt)
- typedef int(∗ **CPLHashSetIterEltFunc**)(void ∗elt, void ∗user_data)

Functions

- CPLHashSet ∗ **CPLHashSetNew** (CPLHashSetHashFunc fnHashFunc, CPLHashSetEqualFunc fnEqualFunc, CPLHashSetFreeEltFunc fnFreeEltFunc)
- void **CPLHashSetDestroy** (CPLHashSet ∗set)
- int **CPLHashSetSize** (const CPLHashSet ∗set)
- void **CPLHashSetForeach** (CPLHashSet ∗set, CPLHashSetIterEltFunc fnIterFunc, void ∗user_data)
- int **CPLHashSetInsert** (CPLHashSet ∗set, void ∗elt)
- void ∗ **CPLHashSetLookup** (CPLHashSet ∗set, const void ∗elt)
- int **CPLHashSetRemove** (CPLHashSet ∗set, const void ∗elt)
- unsigned long **CPLHashSetHashPointer** (const void ∗elt)
- int **CPLHashSetEqualPointer** (const void ∗elt1, const void ∗elt2)
- unsigned long **CPLHashSetHashStr** (const void ∗pszStr)
- int **CPLHashSetEqualStr** (const void ∗pszStr1, const void ∗pszStr2)

44.3.1 Detailed Description

Hash set implementation.

An hash set is a data structure that holds elements that are unique according to a comparison function. Operations on the hash set, such as insertion, removal or lookup, are supposed to be fast if an efficient "hash" function is provided.

44.3.2 Function Documentation

44.3.2.1 void CPLHashSetDestroy (CPLHashSet ∗ *set*)

Destroys an allocated hash set.

This function also frees the elements if a free function was provided at the creation of the hash set.

Parameters:

set the hash set

References _CPLList::pData, and _CPLList::psNext.

44.3.2.2 int CPLHashSetEqualPointer (const void * *elt1*, const void * *elt2*)

Equality function for arbitrary pointers

Parameters:

elt1 the first arbitrary pointer to compare

elt2 the second arbitrary pointer to compare

Returns:

TRUE if the pointers are equal

44.3.2.3 int CPLHashSetEqualStr (const void * *elt1*, const void * *elt2*)

Equality function for strings

Parameters:

elt1 the first string to compare. May be NULL.

elt2 the second string to compare. May be NULL.

Returns:

TRUE if the strings are equal

44.3.2.4 void CPLHashSetForeach (CPLHashSet * *set*, CPLHashSetIterEltFunc *fnIterFunc*, void * *user_data*)

Walk through the hash set and runs the provided function on all the elements

This function is provided the user_data argument of CPLHashSetForeach. It must return TRUE to go on the walk through the hash set, or FALSE to make it stop.

Note : the structure of the hash set must *NOT* be modified during the walk.

Parameters:

set the hash set.

fnIterFunc the function called on each element.

user_data the user data provided to the function.

References _CPLList::pData, and _CPLList::psNext.

44.3.2.5 unsigned long CPLHashSetHashPointer (const void * *elt*)

Hash function for an arbitrary pointer

Parameters:

elt the arbitrary pointer to hash

Returns:

the hash value of the pointer

44.3.2.6 unsigned long CPLHashSetHashStr (const void * *elt*)

Hash function for a zero-terminated string

Parameters:

elt the string to hash. May be NULL.

Returns:

the hash value of the string

44.3.2.7 int CPLHashSetInsert (CPLHashSet * *set*, void * *elt*)

Inserts an element into a hash set.

If the element was already inserted in the hash set, the previous element is replaced by the new element. If a free function was provided, it is used to free the previously inserted element

Parameters:

set the hash set

elt the new element to insert in the hash set

Returns:

TRUE if the element was not already in the hash set

44.3.2.8 void* CPLHashSetLookup (CPLHashSet * *set*, const void * *elt*)

Returns the element found in the hash set corresponding to the element to look up The element must not be modified.

Parameters:

set the hash set

elt the element to look up in the hash set

Returns:

the element found in the hash set or NULL

44.3.2.9 CPLHashSet* CPLHashSetNew (CPLHashSetHashFunc *fnHashFunc*, CPLHashSetEqualFunc *fnEqualFunc*, CPLHashSetFreeEltFunc *fnFreeEltFunc*)

Creates a new hash set

The hash function must return a hash value for the elements to insert. If fnHashFunc is NULL, CPLHashSetHashPointer will be used.

The equal function must return if two elements are equal. If fnEqualFunc is NULL, CPLHashSetEqualPointer will be used.

The free function is used to free elements inserted in the hash set, when the hash set is destroyed, when elements are removed or replaced. If fnFreeEltFunc is NULL, elements inserted into the hash set will not be freed.

Parameters:

fnHashFunc hash function. May be NULL.

fnEqualFunc equal function. May be NULL.

fnFreeEltFunc element free function. May be NULL.

Returns:

a new hash set

44.3.2.10 int CPLHashSetRemove (CPLHashSet * *set*, const void * *elt*)

Removes an element from a hash set

Parameters:

set the hash set

elt the new element to remove from the hash set

Returns:

TRUE if the element was in the hash set

References _CPLList::pData, and _CPLList::psNext.

44.3.2.11 int CPLHashSetSize (const CPLHashSet * *set*)

Returns the number of elements inserted in the hash set

Note: this is not the internal size of the hash set

Parameters:

set the hash set

Returns:

the number of elements in the hash set

44.4 cpl_list.h File Reference

```
#include "cpl_port.h"
```

Classes

- struct **_CPLList**

Typedefs

- typedef struct **_CPLList CPLList**

Functions

- **CPLList** ∗ **CPLListAppend** (**CPLList** ∗psList, void ∗pData)
- **CPLList** ∗ **CPLListInsert** (**CPLList** ∗psList, void ∗pData, int nPosition)
- **CPLList** ∗ **CPLListGetLast** (**CPLList** ∗psList)
- **CPLList** ∗ **CPLListGet** (**CPLList** ∗psList, int nPosition)
- int **CPLListCount** (**CPLList** ∗psList)
- **CPLList** ∗ **CPLListRemove** (**CPLList** ∗psList, int nPosition)
- void **CPLListDestroy** (**CPLList** ∗psList)
- **CPLList** ∗ **CPLListGetNext** (**CPLList** ∗psElement)
- void ∗ **CPLListGetData** (**CPLList** ∗psElement)

44.4.1 Detailed Description

Simplest list implementation. List contains only pointers to stored objects, not objects itself. All operations regarding allocation and freeing memory for objects should be performed by the caller.

44.4.2 Typedef Documentation

44.4.2.1 typedef struct _CPLList CPLList

List element structure.

44.4.3 Function Documentation

44.4.3.1 CPLList∗ CPLListAppend (CPLList ∗ *psList*, void ∗ *pData*)

Append an object list and return a pointer to the modified list. If the input list is NULL, then a new list is created.

Parameters:

psList pointer to list head.

pData pointer to inserted data object. May be NULL.

Returns:

pointer to the head of modified list.

References _CPLList::pData, and _CPLList::psNext.

44.4.3.2 int CPLListCount (CPLList * *psList*)

Return the number of elements in a list.

Parameters:

psList pointer to list head.

Returns:

number of elements in a list.

References _CPLList::psNext.

44.4.3.3 void CPLListDestroy (CPLList * *psList*)

Destroy a list. Caller responsible for freeing data objects contained in list elements.

Parameters:

psList pointer to list head.

References _CPLList::psNext.

44.4.3.4 CPLList* CPLListGet (CPLList * *psList*, int *nPosition*)

Return the pointer to the specified element in a list.

Parameters:

psList pointer to list head.
nPosition the index of the element in the list, 0 being the first element

Returns:

pointer to the specified element in a list.

References _CPLList::psNext.

44.4.3.5 void* CPLListGetData (CPLList * *psElement*)

Return pointer to the data object contained in given list element.

Parameters:

psElement pointer to list element.

Returns:

pointer to the data object contained in given list element.

References _CPLList::pData.

44.4.3.6 CPLList* CPLListGetLast (CPLList * *psList*)

Return the pointer to last element in a list.

Parameters:

psList pointer to list head.

Returns:

pointer to last element in a list.

References _CPLList::psNext.

44.4.3.7 CPLList* CPLListGetNext (CPLList * *psElement*)

Return the pointer to next element in a list.

Parameters:

psElement pointer to list element.

Returns:

pointer to the list element preceded by the given element.

References _CPLList::psNext.

44.4.3.8 CPLList* CPLListInsert (CPLList * *psList*, void * *pData*, int *nPosition*)

Insert an object into list at specified position (zero based). If the input list is NULL, then a new list is created.

Parameters:

psList pointer to list head.

pData pointer to inserted data object. May be NULL.

nPosition position number to insert an object.

Returns:

pointer to the head of modified list.

References _CPLList::pData, and _CPLList::psNext.

44.4.3.9 CPLList* CPLListRemove (CPLList * *psList*, int *nPosition*)

Remove the element from the specified position (zero based) in a list. Data object contained in removed element must be freed by the caller first.

Parameters:

psList pointer to list head.

nPosition position number to delet an element.

Returns:

pointer to the head of modified list.

References _CPLList::psNext.

44.5 cpl_minixml.h File Reference

```
#include "cpl_port.h"
```

Classes

- struct **CPLXMLNode**

Enumerations

- enum **CPLXMLNodeType** {
  CXT_Element = 0, **CXT_Text** = 1, **CXT_Attribute** = 2, **CXT_Comment** = 3,
  CXT_Literal = 4 }

Functions

- **CPLXMLNode** * **CPLParseXMLString** (const char *)
  Parse an XML string into tree form.
- void **CPLDestroyXMLNode** (**CPLXMLNode** *)
  Destroy a tree.
- **CPLXMLNode** * **CPLGetXMLNode** (**CPLXMLNode** *poRoot, const char *pszPath)
  Find node by path.
- **CPLXMLNode** * **CPLSearchXMLNode** (**CPLXMLNode** *poRoot, const char *pszTarget)
  Search for a node in document.
- const char * **CPLGetXMLValue** (**CPLXMLNode** *poRoot, const char *pszPath, const char *pszDefault)
  Fetch element/attribute value.
- **CPLXMLNode** * **CPLCreateXMLNode** (**CPLXMLNode** *poParent, **CPLXMLNodeType** eType, const char *pszText)
  Create an document tree item.
- char * **CPLSerializeXMLTree** (**CPLXMLNode** *psNode)
  Convert tree into string document.
- void **CPLAddXMLChild** (**CPLXMLNode** *psParent, **CPLXMLNode** *psChild)
  Add child node to parent.
- int **CPLRemoveXMLChild** (**CPLXMLNode** *psParent, **CPLXMLNode** *psChild)
  Remove child node from parent.
- void **CPLAddXMLSibling** (**CPLXMLNode** *psOlderSibling, **CPLXMLNode** *psNewSibling)
  Add new sibling.

- **CPLXMLNode** ∗ **CPLCreateXMLElementAndValue** (**CPLXMLNode** ∗psParent, const char ∗pszName, const char ∗pszValue)
  Create an element and text value.
- **CPLXMLNode** ∗ **CPLCloneXMLTree** (**CPLXMLNode** ∗psTree)
  Copy tree.
- int **CPLSetXMLValue** (**CPLXMLNode** ∗psRoot, const char ∗pszPath, const char ∗pszValue)
  Set element value by path.
- void **CPLStripXMLNamespace** (**CPLXMLNode** ∗psRoot, const char ∗pszNameSpace, int bRecurse)
  Strip indicated namespaces.
- void **CPLCleanXMLElementName** (char ∗)
  Make string into safe XML token.
- **CPLXMLNode** ∗ **CPLParseXMLFile** (const char ∗pszFilename)
  Parse XML file into tree.
- int **CPLSerializeXMLTreeToFile** (**CPLXMLNode** ∗psTree, const char ∗pszFilename)
  Write document tree to a file.

44.5.1 Detailed Description

Definitions for CPL mini XML Parser/Serializer.

44.5.2 Enumeration Type Documentation

44.5.2.1 enum CPLXMLNodeType

Enumerator:

CXT_Element Node is an element
CXT_Text Node is a raw text value
CXT_Attribute Node is attribute
CXT_Comment Node is an XML comment.
CXT_Literal Node is a special literal

44.5.3 Function Documentation

44.5.3.1 void CPLAddXMLChild (CPLXMLNode ∗ *psParent*, CPLXMLNode ∗ *psChild*)

Add child node to parent.

The passed child is added to the list of children of the indicated parent. Normally the child is added at the end of the parents child list, but attributes (CXT_Attribute) will be inserted after any other attributes but before any other element type. Ownership of the child node is effectively assumed by the parent node. If the child has siblings (it's psNext is not NULL) they will be trimmed, but if the child has children they are carried with it.

Parameters:

psParent the node to attach the child to. May not be NULL.

psChild the child to add to the parent. May not be NULL. Should not be a child of any other parent.

References CXT_Attribute, CPLXMLNode::eType, CPLXMLNode::psChild, and CPLXMLNode::psNext.

44.5.3.2 void CPLAddXMLSibling (CPLXMLNode * *psOlderSibling*, CPLXMLNode * *psNewSibling*)

Add new sibling.

The passed psNewSibling is added to the end of siblings of the psOlderSibling node. That is, it is added to the end of the psNext chain. There is no special handling if psNewSibling is an attribute. If this is required, use **CPLAddXMLChild()** (p. **??**).

Parameters:

psOlderSibling the node to attach the sibling after.

psNewSibling the node to add at the end of psOlderSiblings psNext chain.

References CPLXMLNode::psNext.

44.5.3.3 void CPLCleanXMLElementName (char * *pszTarget*)

Make string into safe XML token.

Modifiies a string in place to try and make it into a legal XML token that can be used as an element name. This is accomplished by changing any characters not legal in a token into an underscore.

NOTE: This function should implement the rules in section 2.3 of `http://www.w3.org/TR/xml11/` but it doesn't yet do that properly. We only do a rough approximation of that.

Parameters:

pszTarget the string to be adjusted. It is altered in place.

44.5.3.4 CPLXMLNode* CPLCloneXMLTree (CPLXMLNode * *psTree*)

Copy tree.

Creates a deep copy of a **CPLXMLNode** (p. **??**) tree.

Parameters:

psTree the tree to duplicate.

Returns:

a copy of the whole tree.

References CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

44.5.3.5 CPLXMLNode* CPLCreateXMLElementAndValue (CPLXMLNode * *psParent*, const char * *pszName*, const char * *pszValue*)

Create an element and text value.

This is function is a convenient short form for:

```
CPLXMLNode *psTextNode;
CPLXMLNode *psElementNode;

psElementNode = CPLCreateXMLNode( psParent, CXT_Element, pszName );
psTextNode = CPLCreateXMLNode( psElementNode, CXT_Text, pszValue );

return psElementNode;
```

It creates a CXT_Element node, with a CXT_Text child, and attaches the element to the passed parent.

Parameters:

psParent the parent node to which the resulting node should be attached. May be NULL to keep as freestanding.

pszName the element name to create.

pszValue the text to attach to the element. Must not be NULL.

Returns:

the pointer to the new element node.

References CXT_Element, and CXT_Text.

44.5.3.6 CPLXMLNode* CPLCreateXMLNode (CPLXMLNode * *poParent*, CPLXMLNodeType *eType*, const char * *pszText*)

Create an document tree item.

Create a single **CPLXMLNode** (p. **??**) object with the desired value and type, and attach it as a child of the indicated parent.

Parameters:

poParent the parent to which this node should be attached as a child. May be NULL to keep as free standing.

eType the type of the newly created node

pszText the value of the newly created node

Returns:

the newly created node, now owned by the caller (or parent node).

References CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

44.5.3.7 void CPLDestroyXMLNode (CPLXMLNode * *psNode*)

Destroy a tree.

This function frees resources associated with a **CPLXMLNode** (p. **??**) and all its children nodes.

Parameters:

psNode the tree to free.

References CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

44.5.3.8 CPLXMLNode* CPLGetXMLNode (CPLXMLNode * *psRoot*, const char * *pszPath*)

Find node by path.

Searches the document or subdocument indicated by psRoot for an element (or attribute) with the given path. The path should consist of a set of element names separated by dots, not including the name of the root element (psRoot). If the requested element is not found NULL is returned.

Attribute names may only appear as the last item in the path.

The search is done from the root nodes children, but all intermediate nodes in the path must be specified. Seaching for "name" would only find a name element or attribute if it is a direct child of the root, not at any level in the subdocument.

If the pszPath is prefixed by "=" then the search will begin with the root node, and it's siblings, instead of the root nodes children. This is particularly useful when searching within a whole document which is often prefixed by one or more "junk" nodes like the <?xml> declaration.

Parameters:

psRoot the subtree in which to search. This should be a node of type CXT_Element. NULL is safe.

pszPath the list of element names in the path (dot separated).

Returns:

the requested element node, or NULL if not found.

References CXT_Text, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

44.5.3.9 const char* CPLGetXMLValue (CPLXMLNode * *psRoot*, const char * *pszPath*, const char * *pszDefault*)

Fetch element/attribute value.

Searches the document for the element/attribute value associated with the path. The corresponding node is internally found with **CPLGetXMLNode()** (p. **??**) (see there for details on path handling). Once found, the value is considered to be the first CXT_Text child of the node.

If the attribute/element search fails, or if the found node has not value then the passed default value is returned.

The returned value points to memory within the document tree, and should not be altered or freed.

Parameters:

psRoot the subtree in which to search. This should be a node of type CXT_Element. NULL is safe.

pszPath the list of element names in the path (dot separated). An empty path means get the value of the psRoot node.

pszDefault the value to return if a corresponding value is not found, may be NULL.

Returns:

the requested value or pszDefault if not found.

References CXT_Attribute, CXT_Element, CXT_Text, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

44.5.3.10 CPLXMLNode* CPLParseXMLFile (const char * *pszFilename*)

Parse XML file into tree.

The named file is opened, loaded into memory as a big string, and parsed with **CPLParseXMLString()** (p. **??**). Errors in reading the file or parsing the XML will be reported by CPLError().

The "large file" API is used, so XML files can come from virtualized files.

Parameters:

pszFilename the file to open.

Returns:

NULL on failure, or the document tree on success.

References VSIFCloseL(), VSIFOpenL(), VSIFReadL(), VSIFSeekL(), and VSIFTellL().

44.5.3.11 CPLXMLNode* CPLParseXMLString (const char * *pszString*)

Parse an XML string into tree form.

The passed document is parsed into a **CPLXMLNode** (p. **??**) tree representation. If the document is not well formed XML then NULL is returned, and errors are reported via CPLError(). No validation beyond wellformedness is done. The **CPLParseXMLFile()** (p. **??**) convenience function can be used to parse from a file.

The returned document tree is is owned by the caller and should be freed with **CPLDestroyXMLNode()** (p. **??**) when no longer needed.

If the document has more than one "root level" element then those after the first will be attached to the first as siblings (via the psNext pointers) even though there is no common parent. A document with no XML structure (no angle brackets for instance) would be considered well formed, and returned as a single CXT_Text node.

Parameters:

pszString the document to parse.

Returns:

parsed tree or NULL on error.

References CXT_Attribute, CXT_Comment, CXT_Element, CXT_Literal, CXT_Text, and CPLXMLNode::pszValue.

44.5.3.12 int CPLRemoveXMLChild (CPLXMLNode * *psParent*, CPLXMLNode * *psChild*)

Remove child node from parent.

The passed child is removed from the child list of the passed parent, but the child is not destroyed. The child retains ownership of it's own children, but is cleanly removed from the child list of the parent.

Parameters:

psParent the node to the child is attached to.

psChild the child to remove.

Returns:

TRUE on success or FALSE if the child was not found.

References CPLXMLNode::psChild, and CPLXMLNode::psNext.

44.5.3.13 CPLXMLNode* CPLSearchXMLNode (CPLXMLNode * *psRoot*, const char * *pszElement*)

Search for a node in document.

Searches the children (and potentially siblings) of the documented passed in for the named element or attribute. To search following siblings as well as children, prefix the pszElement name with an equal sign. This function does an in-order traversal of the document tree. So it will first match against the current node, then it's first child, that childs first child, and so on.

Use **CPLGetXMLNode()** (p. **??**) to find a specific child, or along a specific node path.

Parameters:

psRoot the subtree to search. This should be a node of type CXT_Element. NULL is safe.

pszElement the name of the element or attribute to search for.

Returns:

The matching node or NULL on failure.

References CXT_Attribute, CXT_Element, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

44.5.3.14 char* CPLSerializeXMLTree (CPLXMLNode * *psNode*)

Convert tree into string document.

This function converts a **CPLXMLNode** (p. **??**) tree representation of a document into a flat string representation. White space indentation is used visually preserve the tree structure of the document. The returned document becomes owned by the caller and should be freed with CPLFree() when no longer needed.

Parameters:

psNode

Returns:

the document on success or NULL on failure.

References CPLXMLNode::psNext.

44.5.3.15 int CPLSerializeXMLTreeToFile (CPLXMLNode * *psTree*, const char * *pszFilename*)

Write document tree to a file.

The passed document tree is converted into one big string (with **CPLSerializeXMLTree()** (p. **??**)) and then written to the named file. Errors writing the file will be reported by CPLError(). The source document tree is not altered. If the output file already exists it will be overwritten.

Parameters:

psTree the document tree to write.

pszFilename the name of the file to write to.

Returns:

TRUE on success, FALSE otherwise.

References VSIFCloseL(), VSIFOpenL(), and VSIFWriteL().

44.5.3.16 int CPLSetXMLValue (CPLXMLNode * *psRoot*, const char * *pszPath*, const char * *pszValue*)

Set element value by path.

Find (or create) the target element or attribute specified in the path, and assign it the indicated value.

Any path elements that do not already exist will be created. The target nodes value (the first CXT_Text child) will be replaced with the provided value.

If the target node is an attribute instead of an element, the name should be prefixed with a #.

Example: CPLSetXMLValue("Citation.Id.Description", "DOQ dataset"); CPLSetXMLValue("Citation.Id.Description.#name", "doq");

Parameters:

psRoot the subdocument to be updated.

pszPath the dot seperated path to the target element/attribute.

pszValue the text value to assign.

Returns:

TRUE on success.

References CXT_Attribute, CXT_Element, CXT_Text, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

44.5.3.17 void CPLStripXMLNamespace (CPLXMLNode * *psRoot*, const char * *pszNamespace*, int *bRecurse*)

Strip indicated namespaces.

The subdocument (psRoot) is recursively examined, and any elements with the indicated namespace prefix will have the namespace prefix stripped from the element names. If the passed namespace is NULL, then all namespace prefixes will be stripped.

Nodes other than elements should remain unaffected. The changes are made "in place", and should not alter any node locations, only the pszValue field of affected nodes.

Parameters:

psRoot the document to operate on.

pszNamespace the name space prefix (not including colon), or NULL.

bRecurse TRUE to recurse over whole document, or FALSE to only operate on the passed node.

References CXT_Attribute, CXT_Element, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

44.6 cpl_odbc.h File Reference

```
#include "cpl_port.h"
#include <sql.h>
#include <sqlext.h>
#include <odbcinst.h>
#include "cpl_string.h"
```

Classes

- class **CPLODBCDriverInstaller**
- class **CPLODBCSession**
- class **CPLODBCStatement**

Defines

- #define **ODBC_FILENAME_MAX** (255 + 1)
- #define **_SQLULEN** SQLULEN
- #define **_SQLLEN** SQLLEN

44.6.1 Detailed Description

ODBC Abstraction Layer (C++).

44.7 cpl_port.h File Reference

```
#include "cpl_config.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdarg.h>
#include <string.h>
#include <ctype.h>
#include <limits.h>
#include <time.h>
#include <errno.h>
#include <locale.h>
```

Defines

- #define **CPL_FRMT_GIB** "%lld"
- #define **CPL_FRMT_GUIB** "%llu"
- #define **CPL_C_START** extern "C" {
- #define **CPL_C_END** }
- #define **CPL_ODLL**
- #define **FORCE_CDECL**
- #define **CPL_INLINE**
- #define **NULL** 0
- #define **FALSE** 0
- #define **TRUE** 1
- #define **MIN**(a, b) ((a<b) ? a : b)
- #define **MAX**(a, b) ((a>b) ? a : b)
- #define **ABS**(x) ((x<0) ? (-1∗(x)) : x)
- #define **CPLIsEqual**(x, y) (fabs((x) - (y)) < 0.0000000000001)
- #define **EQUALN**(a, b, n) (strncasecmp(a,b,n)==0)
- #define **EQUAL**(a, b) (strcasecmp(a,b)==0)
- #define **CPLIsNan**(x) isnan(x)
- #define **CPLIsInf**(x) FALSE
- #define **CPLIsFinite**(x) (!isnan(x))
- #define **CPL_MSB**
- #define **CPL_IS_LSB** 0
- #define **CPL_SWAP16**(x)
- #define **CPL_SWAP16PTR**(x)
- #define **CPL_SWAP32**(x)
- #define **CPL_SWAP32PTR**(x)
- #define **CPL_SWAP64PTR**(x)
- #define **CPL_SWAPDOUBLE**(p) CPL_SWAP64PTR(p)
- #define **CPL_MSBWORD16**(x) (x)
- #define **CPL_LSBWORD16**(x) CPL_SWAP16(x)
- #define **CPL_MSBWORD32**(x) (x)

- #define **CPL_LSBWORD32**(x) CPL_SWAP32(x)
- #define **CPL_MSBPTR16**(x)
- #define **CPL_LSBPTR16**(x) CPL_SWAP16PTR(x)
- #define **CPL_MSBPTR32**(x)
- #define **CPL_LSBPTR32**(x) CPL_SWAP32PTR(x)
- #define **CPL_MSBPTR64**(x)
- #define **CPL_LSBPTR64**(x) CPL_SWAP64PTR(x)
- #define **CPL_LSBINT16PTR**(x) ((∗(GByte∗)(x)) | ((∗(GByte∗)((x)+1)) << 8))
- #define **CPL_LSBINT32PTR**(x)
- #define **UNREFERENCED_PARAM**(param) ((void)param)
- #define **CPL_CVSID**(string)
- #define **CPL_PRINT_FUNC_FORMAT**(format_idx, arg_idx)

Typedefs

- typedef int **GInt32**
- typedef unsigned int **GUInt32**
- typedef short **GInt16**
- typedef unsigned short **GUInt16**
- typedef unsigned char **GByte**
- typedef int **GBool**
- typedef long long **GIntBig**
- typedef unsigned long long **GUIntBig**

44.7.1 Detailed Description

Core portability definitions for CPL.

44.7.2 Define Documentation

44.7.2.1 #define CPL_CVSID(string)

Value:

```
static char cpl_cvsid[] = string; \
static char *cvsid_aw() { return( cvsid_aw() ? ((char *) NULL) : cpl_cvsid ); }
```

44.7.2.2 #define CPL_LSBINT16PTR(x) ((∗(GByte∗)(x)) | ((∗(GByte∗)((x)+1)) << 8))

Return a Int16 from the 2 bytes ordered in LSB order at address x

44.7.2.3 #define CPL_LSBINT32PTR(x)

Value:

```
((*(GByte*)(x)) | ((*(GByte*)((x)+1)) << 8) | \
                                ((*(GByte*)((x)+2)) << 16) | ((*(GByte*)((x)+3)) << 24))
```

Return a Int32 from the 4 bytes ordered in LSB order at address x

44.7.2.4 #define CPL_SWAP16(x)

Value:

```
((GUInt16)( \
            (((GUInt16)(x) & 0x00ffU) << 8) | \
            (((GUInt16)(x) & 0xff00U) >> 8) ))
```

44.7.2.5 #define CPL_SWAP16PTR(x)

Value:

```
{                                                               \
    GByte       byTemp, *_pabyDataT = (GByte *) (x);            \
                                                                \
    byTemp = _pabyDataT[0];                                     \
    _pabyDataT[0] = _pabyDataT[1];                              \
    _pabyDataT[1] = byTemp;                                     \
}
```

44.7.2.6 #define CPL_SWAP32(x)

Value:

```
((GUInt32)( \
            (((GUInt32)(x) & (GUInt32)0x000000ffUL) << 24) | \
            (((GUInt32)(x) & (GUInt32)0x0000ff00UL) <<  8) | \
            (((GUInt32)(x) & (GUInt32)0x00ff0000UL) >>  8) | \
            (((GUInt32)(x) & (GUInt32)0xff000000UL) >> 24) ))
```

44.7.2.7 #define CPL_SWAP32PTR(x)

Value:

```
{                                                               \
    GByte       byTemp, *_pabyDataT = (GByte *) (x);            \
                                                                \
    byTemp = _pabyDataT[0];                                     \
    _pabyDataT[0] = _pabyDataT[3];                              \
    _pabyDataT[3] = byTemp;                                     \
    byTemp = _pabyDataT[1];                                     \
    _pabyDataT[1] = _pabyDataT[2];                              \
    _pabyDataT[2] = byTemp;                                     \
}
```

44.7.2.8 #define CPL_SWAP64PTR(x)

Value:

```
{                                                               \
    GByte       byTemp, *_pabyDataT = (GByte *) (x);            \
                                                                \
```

```
    byTemp = _pabyDataT[0];                                 \
    _pabyDataT[0] = _pabyDataT[7];                          \
    _pabyDataT[7] = byTemp;                                 \
    byTemp = _pabyDataT[1];                                 \
    _pabyDataT[1] = _pabyDataT[6];                          \
    _pabyDataT[6] = byTemp;                                 \
    byTemp = _pabyDataT[2];                                 \
    _pabyDataT[2] = _pabyDataT[5];                          \
    _pabyDataT[5] = byTemp;                                 \
    byTemp = _pabyDataT[3];                                 \
    _pabyDataT[3] = _pabyDataT[4];                          \
    _pabyDataT[4] = byTemp;                                 \
}
```

44.8 cpl_quad_tree.h File Reference

```
#include "cpl_port.h"
```

Classes

- struct **CPLRectObj**

Typedefs

- typedef struct _CPLQuadTree **CPLQuadTree**
- typedef void(∗ **CPLQuadTreeGetBoundsFunc**)(const void ∗hFeature, CPLRectObj ∗pBounds)
- typedef int(∗ **CPLQuadTreeForeachFunc**)(void ∗pElt, void ∗pUserData)
- typedef void(∗ **CPLQuadTreeDumpFeatureFunc**)(const void ∗hFeature, int nIndentLevel, void ∗pUserData)

Functions

- CPLQuadTree ∗ **CPLQuadTreeCreate** (const CPLRectObj ∗pGlobalBounds, CPLQuadTreeGetBoundsFunc pfnGetBounds)
- void **CPLQuadTreeDestroy** (CPLQuadTree ∗hQuadtree)
- void **CPLQuadTreeSetBucketCapacity** (CPLQuadTree ∗hQuadtree, int nBucketCapacity)
- int **CPLQuadTreeGetAdvisedMaxDepth** (int nExpectedFeatures)
- void **CPLQuadTreeSetMaxDepth** (CPLQuadTree ∗hQuadtree, int nMaxDepth)
- void **CPLQuadTreeInsert** (CPLQuadTree ∗hQuadtree, void ∗hFeature)
- void ∗∗ **CPLQuadTreeSearch** (const CPLQuadTree ∗hQuadtree, const CPLRectObj ∗pAoi, int ∗pnFeatureCount)
- void **CPLQuadTreeForeach** (const CPLQuadTree ∗hQuadtree, CPLQuadTreeForeachFunc pfnForeach, void ∗pUserData)
- void **CPLQuadTreeDump** (const CPLQuadTree ∗hQuadtree, CPLQuadTreeDumpFeatureFunc pfnDumpFeatureFunc, void ∗pUserData)
- void **CPLQuadTreeGetStats** (const CPLQuadTree ∗hQuadtree, int ∗pnFeatureCount, int ∗pnNodeCount, int ∗pnMaxDepth, int ∗pnMaxBucketCapacity)

44.8.1 Detailed Description

Quad tree implementation.

A quadtree is a tree data structure in which each internal node has up to four children. Quadtrees are most often used to partition a two dimensional space by recursively subdividing it into four quadrants or regions

44.8.2 Function Documentation

44.8.2.1 CPLQuadTree∗ CPLQuadTreeCreate (const CPLRectObj ∗ *pGlobalBounds*, CPLQuadTreeGetBoundsFunc *pfnGetBounds*)

Create a new quadtree

Parameters:

pGlobalBounds a pointer to the global extent of all the elements that will be inserted

pfnGetBounds a user provided function to get the bounding box of the inserted elements

Returns:

a newly allocated quadtree

44.8.2.2 void CPLQuadTreeDestroy (CPLQuadTree * *hQuadTree*)

Destroy a quadtree

Parameters:

hQuadTree the quad tree to destroy

44.8.2.3 void CPLQuadTreeForeach (const CPLQuadTree * *hQuadTree*, CPLQuadTreeForeachFunc *pfnForeach*, void * *pUserData*)

Walk through the quadtree and runs the provided function on all the elements

This function is provided with the user_data argument of pfnForeach. It must return TRUE to go on the walk through the hash set, or FALSE to make it stop.

Note : the structure of the quadtree must *NOT* be modified during the walk.

Parameters:

hQuadTree the quad tree

pfnForeach the function called on each element.

pUserData the user data provided to the function.

44.8.2.4 int CPLQuadTreeGetAdvisedMaxDepth (int *nExpectedFeatures*)

Returns the optimal depth of a quadtree to hold nExpectedFeatures

Parameters:

nExpectedFeatures the expected maximum number of elements to be inserted

Returns:

the optimal depth of a quadtree to hold nExpectedFeatures

44.8.2.5 void CPLQuadTreeInsert (CPLQuadTree * *hQuadTree*, void * *hFeature*)

Insert a feature into a quadtree

Parameters:

hQuadTree the quad tree

hFeature the feature to insert

44.8.2.6 void** CPLQuadTreeSearch (const CPLQuadTree * *hQuadTree*, const CPLRectObj * *pAoi*, int * *pnFeatureCount*)

Returns all the elements inserted whose bounding box intersects the provided area of interest

Parameters:

hQuadTree the quad tree

pAoi the pointer to the area of interest

pnFeatureCount the user data provided to the function.

Returns:

an array of features that must be freed with CPLFree

44.8.2.7 void CPLQuadTreeSetBucketCapacity (CPLQuadTree * *hQuadTree*, int *nBucketCapacity*)

Set the maximum capacity of a node of a quadtree. The default value is 8. Note that the maximum capacity will only be honoured if the features inserted have a point geometry. Otherwise it may be exceeded.

Parameters:

hQuadTree the quad tree

nBucketCapacity the maximum capactiy of a node of a quadtree

44.8.2.8 void CPLQuadTreeSetMaxDepth (CPLQuadTree * *hQuadTree*, int *nMaxDepth*)

Set the maximum depth of a quadtree. By default, quad trees have no maximum depth, but a maximum bucket capacity.

Parameters:

hQuadTree the quad tree

nMaxDepth the maximum depth allowed

44.9 cpl_string.h File Reference

```
#include "cpl_vsi.h"
#include "cpl_error.h"
#include "cpl_conv.h"
#include <string>
```

Classes

- class **CPLString**

Defines

- #define **CSLT_HONOURSTRINGS** 0x0001
- #define **CSLT_ALLOWEMPTYTOKENS** 0x0002
- #define **CSLT_PRESERVEQUOTES** 0x0004
- #define **CSLT_PRESERVEESCAPES** 0x0008
- #define **CSLT_STRIPLEADSPACES** 0x0010
- #define **CSLT_STRIPENDSPACES** 0x0020
- #define **CPLES_BackslashQuotable** 0
- #define **CPLES_XML** 1
- #define **CPLES_URL** 2
- #define **CPLES_SQL** 3
- #define **CPLES_CSV** 4
- #define **CPL_ENC_LOCALE** ""
- #define **CPL_ENC_UTF8** "UTF-8"
- #define **CPL_ENC_UTF16** "UTF-16"
- #define **CPL_ENC_UCS2** "UCS-2"
- #define **CPL_ENC_UCS4** "UCS-4"
- #define **CPL_ENC_ASCII** "ASCII"
- #define **CPL_ENC_ISO8859_1** "ISO-8859-1"
- #define **std_string** std::string

Enumerations

- enum **CPLValueType** { **CPL_VALUE_STRING**, **CPL_VALUE_REAL**, **CPL_VALUE_-INTEGER** }

Functions

- char ** **CSLAddString** (char **papszStrList, const char *pszNewString)
- int **CSLCount** (char **papszStrList)
- const char * **CSLGetField** (char **, int)
- void **CSLDestroy** (char **papszStrList)
- char ** **CSLDuplicate** (char **papszStrList)
- char ** **CSLMerge** (char **papszOrig, char **papszOverride)

  Merge two lists.

- char ** **CSLTokenizeString** (const char *pszString)
- char ** **CSLTokenizeStringComplex** (const char *pszString, const char *pszDelimiter, int bHonourStrings, int bAllowEmptyTokens)
- char ** **CSLTokenizeString2** (const char *pszString, const char *pszDelimeter, int nCSLTFlags)
- int **CSLPrint** (char **papszStrList, FILE *fpOut)
- char ** **CSLLoad** (const char *pszFname)
- int **CSLSave** (char **papszStrList, const char *pszFname)
- char ** **CSLInsertStrings** (char **papszStrList, int nInsertAtLineNo, char **papszNewLines)
- char ** **CSLInsertString** (char **papszStrList, int nInsertAtLineNo, const char *pszNewLine)
- char ** **CSLRemoveStrings** (char **papszStrList, int nFirstLineToDelete, int nNumToRemove, char ***ppapszRetStrings)
- int **CSLFindString** (char **, const char *)
- int **CSLPartialFindString** (char **papszHaystack, const char *pszNeedle)
- int **CSLFindName** (char **papszStrList, const char *pszName)
- int **CSLTestBoolean** (const char *pszValue)
- int **CSLFetchBoolean** (char **papszStrList, const char *pszKey, int bDefault)
- const char * **CPLSPrintf** (const char *fmt,...) CPL_PRINT_FUNC_FORMAT(1
- const char *char ** **CSLAppendPrintf** (char **papszStrList, const char *fmt,...) CPL_PRINT_-FUNC_FORMAT(2
- const char *char **int **CPLVASPrintf** (char **buf, const char *fmt, va_list args)
- const char * **CPLParseNameValue** (const char *pszNameValue, char **ppszKey)
- const char * **CSLFetchNameValue** (char **papszStrList, const char *pszName)
- char ** **CSLFetchNameValueMultiple** (char **papszStrList, const char *pszName)
- char ** **CSLAddNameValue** (char **papszStrList, const char *pszName, const char *pszValue)
- char ** **CSLSetNameValue** (char **papszStrList, const char *pszName, const char *pszValue)
- void **CSLSetNameValueSeparator** (char **papszStrList, const char *pszSeparator)
- char * **CPLEscapeString** (const char *pszString, int nLength, int nScheme)
- char * **CPLUnescapeString** (const char *pszString, int *pnLength, int nScheme)
- char * **CPLBinaryToHex** (int nBytes, const GByte *pabyData)
- GByte * **CPLHexToBinary** (const char *pszHex, int *pnBytes)
- CPLValueType **CPLGetValueType** (const char *pszValue)
- char * **CPLRecode** (const char *pszSource, const char *pszSrcEncoding, const char *pszDstEncoding)
- char * **CPLRecodeFromWChar** (const wchar_t *pwszSource, const char *pszSrcEncoding, const char *pszDstEncoding)
- wchar_t * **CPLRecodeToWChar** (const char *pszSource, const char *pszSrcEncoding, const char *pszDstEncoding)

44.9.1 Detailed Description

Various convenience functions for working with strings and string lists.

A StringList is just an array of strings with the last pointer being NULL. An empty StringList may be either a NULL pointer, or a pointer to a pointer memory location with a NULL value.

A common convention for StringLists is to use them to store name/value lists. In this case the contents are treated like a dictionary of name/value pairs. The actual data is formatted with each string having the format "<name>:<value>" (though "=" is also an acceptable separator). A number of the functions in the file operate on name/value style string lists (such as **CSLSetNameValue()** (p. **??**), and CSLFetch-NameValue()).

44.9.2 Function Documentation

44.9.2.1 char* CPLBinaryToHex (int *nBytes*, const GByte * *pabyData*)

Binary to hexadecimal translation.

Parameters:

nBytes number of bytes of binary data in pabyData.

pabyData array of data bytes to translate.

Returns:

hexadecimal translation, zero terminated. Free with CPLFree().

44.9.2.2 char* CPLEscapeString (const char * *pszInput*, int *nLength*, int *nScheme*)

Apply escaping to string to preserve special characters.

This function will "escape" a variety of special characters to make the string suitable to embed within a string constant or to write within a text stream but in a form that can be reconstitued to it's original form. The escaping will even preserve zero bytes allowing preservation of raw binary data.

CPLES_BackslashQuotable(0): This scheme turns a binary string into a form suitable to be placed within double quotes as a string constant. The backslash, quote, '\0' and newline characters are all escaped in the usual C style.

CPLES_XML(1): This scheme converts the '<', '<' and '&' characters into their XML/HTML equivelent (>, < and &) making a string safe to embed as CDATA within an XML element. The '\0' is not escaped and should not be included in the input.

CPLES_URL(2): Everything except alphanumerics and the underscore are converted to a percent followed by a two digit hex encoding of the character (leading zero supplied if needed). This is the mechanism used for encoding values to be passed in URLs.

CPLES_SQL(3): All single quotes are replaced with two single quotes. Suitable for use when constructing literal values for SQL commands where the literal will be enclosed in single quotes.

CPLES_CSV(4): If the values contains commas, double quotes, or newlines it placed in double quotes, and double quotes in the value are doubled. Suitable for use when constructing field values for .csv files. Note that **CPLUnescapeString()** (p. **??**) currently does not support this format, only **CPLEscapeString()** (p. **??**). See cpl_csv.cpp for csv parsing support.

Parameters:

pszInput the string to escape.

nLength The number of bytes of data to preserve. If this is -1 the strlen(pszString) function will be used to compute the length.

nScheme the encoding scheme to use.

Returns:

an escaped, zero terminated string that should be freed with CPLFree() when no longer needed.

44.9.2.3 CPLValueType CPLGetValueType (const char * *pszValue*)

Detect the type of the value contained in a string, whether it is a real, an integer or a string Leading and trailing spaces are skipped in the analysis.

Parameters:

pszValue the string to analyze

Returns:

returns the type of the value contained in the string.

44.9.2.4 GByte* CPLHexToBinary (const char * *pszHex*, int * *pnBytes*)

Hexadecimal to binary translation

Parameters:

pszHex the input hex encoded string.

pnBytes the returned count of decoded bytes placed here.

Returns:

returns binary buffer of data - free with CPLFree().

44.9.2.5 const char* CPLParseNameValue (const char * *pszNameValue*, char ** *ppszKey*)

Parse NAME=VALUE string into name and value components.

Note that if ppszKey is non-NULL, the key (or name) portion will be allocated using VSIMalloc(), and returned in that pointer. It is the applications responsibility to free this string, but the application should not modify or free the returned value portion.

This function also support "NAME:VALUE" strings and will strip white space from around the delimeter when forming name and value strings.

Eventually CSLFetchNameValue() and friends may be modified to use **CPLParseNameValue()** (p. **??**).

Parameters:

pszNameValue string in "NAME=VALUE" format.

ppszKey optional pointer though which to return the name portion.

Returns:

the value portion (pointing into original string).

44.9.2.6 char* CPLRecodeFromWChar (const wchar_t * *pwszSource*, const char * *pszSrcEncoding*, const char * *pszDstEncoding*)

Convert wchar_t string to UTF-8.

Convert a wchar_t string into a multibyte utf-8 string. The only guaranteed supported source encoding is CPL_ENC_UCS2, and the only guaranteed supported destination encodings are CPL_ENC_UTF8, CPL_-ENC_ASCII and CPL_ENC_ISO8859_1. In some cases (ie. using iconv()) other encodings may also be supported.

Note that the wchar_t type varies in size on different systems. On win32 it is normally 2 bytes, and on unix 4 bytes.

If an error occurs an error may, or may not be posted with CPLError().

Parameters:

pwszSource the source wchar_t string, terminated with a 0 wchar_t.

pszSrcEncoding the source encoding, typically CPL_ENC_UCS2.

pszDstEncoding the destination encoding, typically CPL_ENC_UTF8.

Returns:

a zero terminated multi-byte string which should be freed with CPLFree(), or NULL if an error occurs.

44.9.2.7 wchar_t* CPLRecodeToWChar (const char * *pszSource*, const char * *pszSrcEncoding*, const char * *pszDstEncoding*)

Convert UTF-8 string to a wchar_t string.

Convert a 8bit, multi-byte per character input string into a wide character (wchar_t) string. The only guaranteed supported source encodings are CPL_ENC_UTF8, CPL_ENC_ASCII and CPL_ENC_ISO8869_1 (LATIN1). The only guaranteed supported destination encoding is CPL_ENC_UCS2. Other source and destination encodings may be supported depending on the underlying implementation.

Note that the wchar_t type varies in size on different systems. On win32 it is normally 2 bytes, and on unix 4 bytes.

If an error occurs an error may, or may not be posted with CPLError().

Parameters:

pszSource input multi-byte character string.

pszSrcEncoding source encoding, typically CPL_ENC_UTF8.

pszDstEncoding destination encoding, typically CPL_ENC_UCS2.

Returns:

the zero terminated wchar_t string (to be freed with CPLFree()) or NULL on error.

44.9.2.8 char* CPLUnescapeString (const char * *pszInput*, int * *pnLength*, int *nScheme*)

Unescape a string.

This function does the opposite of **CPLEscapeString()** (p. **??**). Given a string with special values escaped according to some scheme, it will return a new copy of the string returned to it's original form.

Parameters:

pszInput the input string. This is a zero terminated string.

pnLength location to return the length of the unescaped string, which may in some cases include embedded '\0' characters.

nScheme the escaped scheme to undo (see **CPLEscapeString()** (p. **??**) for a list).

Returns:

a copy of the unescaped string that should be freed by the application using CPLFree() when no longer needed.

44.9.2.9 int CSLCount (char ** *papszStrList*)

Return number of items in a string list.

Returns the number of items in a string list, not counting the terminating NULL. Passing in NULL is safe, and will result in a count of zero.

Lists are counted by iterating through them so long lists will take more time than short lists. Care should be taken to avoid using **CSLCount()** (p. **??**) as an end condition for loops as it will result in O(n^2) behavior.

Parameters:

papszStrList the string list to count.

Returns:

the number of entries.

44.9.2.10 void CSLDestroy (char ** *papszStrList*)

Free string list.

Frees the passed string list (null terminated array of strings). It is safe to pass NULL.

Parameters:

papszStrList the list to free.

44.9.2.11 char** CSLDuplicate (char ** *papszStrList*)

Clone a string list.

Efficiently allocates a copy of a string list. The returned list is owned by the caller and should be freed with **CSLDestroy()** (p. **??**).

Parameters:

papszStrList the input string list.

Returns:

newly allocated copy.

44.9.2.12 int CSLFindName (char ** *papszStrList*, const char * *pszName*)

Find StringList entry with given key name.

Parameters:

papszStrList the string list to search.

pszName the key value to look for (case insensitive).

Returns:

-1 on failure or the list index of the first occurance matching the given key.

44.9.2.13 int CSLFindString (char ** *papszList*, const char * *pszTarget*)

Find a string within a string list.

Returns the index of the entry in the string list that contains the target string. The string in the string list must be a full match for the target, but the search is case insensitive.

Parameters:

papszList the string list to be searched.

pszTarget the string to be searched for.

Returns:

the index of the string within the list or -1 on failure.

44.9.2.14 char** CSLLoad (const char * *pszFname*)

Load a text file into a string list.

The VSI*L API is used, so **VSIFOpenL()** (p. **??**) supported objects that aren't physical files can also be accessed. Files are returned as a string list, with one item in the string list per line. End of line markers are stripped (by **CPLReadLineL()** (p. **??**)).

If reading the file fails a CPLError() will be issued and NULL returned.

Parameters:

pszFname the name of the file to read.

Returns:

a string list with the files lines, now owned by caller.

References VSIFCloseL(), VSIFEofL(), and VSIFOpenL().

44.9.2.15 char** CSLMerge (char ** *papszOrig*, char ** *papszOverride*)

Merge two lists.

The two lists are merged, ensuring that if any keys appear in both that the value from the second (papszOverride) list take precidence.

Parameters:

papszOrig the original list, being modified.

papszOverride the list of items being merged in. This list is unaltered and remains owned by the caller.

Returns:

updated list.

44.9.2.16 int CSLPartialFindString (char ** *papszHaystack*, const char * *pszNeedle*)

Find a substring within a string list.

Returns the index of the entry in the string list that contains the target string as a substring. The search is case sensitive (unlike **CSLFindString()** (p. **??**)).

Parameters:

papszHaystack the string list to be searched.

pszNeedle the substring to be searched for.

Returns:

the index of the string within the list or -1 on failure.

44.9.2.17 char** CSLSetNameValue (char ** *papszList*, const char * *pszName*, const char * *pszValue*)

Assign value to name in StringList.

Set the value for a given name in a StringList of "Name=Value" pairs ("Name:Value" pairs are also supported for backward compatibility with older stuff.)

If there is already a value for that name in the list then the value is changed, otherwise a new "Name=Value" pair is added.

Parameters:

papszList the original list, the modified version is returned.

pszName the name to be assigned a value. This should be a well formed token (no spaces or very special characters).

pszValue the value to assign to the name. This should not contain any newlines (CR or LF) but is otherwise pretty much unconstrained. If NULL any corresponding value will be removed.

Returns:

modified stringlist.

44.9.2.18 void CSLSetNameValueSeparator (char ** *papszList*, const char * *pszSeparator*)

Replace the default separator (":" or "=") with the passed separator in the given name/value list.

Note that if a separator other than ":" or "=" is used, the resulting list will not be manipulatable by the CSL name/value functions any more.

The **CPLParseNameValue()** (p. **??**) function is used to break the existing lines, and it also strips white space from around the existing delimiter, thus the old separator, and any white space will be replaced by the new separator. For formatting purposes it may be desireable to include some white space in the new separator. eg. ": " or " = ".

Parameters:

papszList the list to update. Component strings may be freed but the list array will remain at the same location.

pszSeparator the new separator string to insert.

44.9.2.19 int CSLTestBoolean (const char * *pszValue*)

Test what boolean value contained in the string.

If pszValue is "NO", "FALSE", "OFF" or "0" will be returned FALSE. Otherwise, TRUE will be returned.

Parameters:

pszValue the string should be tested.

Returns:

TRUE or FALSE.

44.9.2.20 char** CSLTokenizeString2 (const char * *pszString*, const char * *pszDelimiters*, int *nCSLTFlags*)

Tokenize a string.

This function will split a string into tokens based on specified' delimeter(s) with a variety of options. The returned result is a string list that should be freed with **CSLDestroy()** (p. **??**) when no longer needed.

The available parsing options are:

- CSLT_ALLOWEMPTYTOKENS: allow the return of empty tokens when two delimiters in a row occur with no other text between them. If not set, empty tokens will be discarded;
- CSLT_STRIPLEADSPACES: strip leading space characters from the token (as reported by isspace());
- CSLT_STRIPENDSPACES: strip ending space characters from the token (as reported by isspace());
- CSLT_HONOURSTRINGS: double quotes can be used to hold values that should not be broken into multiple tokens;
- CSLT_PRESERVEQUOTES: string quotes are carried into the tokens when this is set, otherwise they are removed;
- CSLT_PRESERVEESCAPES: if set backslash escapes (for backslash itself, and for literal double quotes) will be preserved in the tokens, otherwise the backslashes will be removed in processing.

Example:

Parse a string into tokens based on various white space (space, newline, tab) and then print out results and cleanup. Quotes may be used to hold white space in tokens.

```
char **papszTokens;
int i;

papszTokens =
    CSLTokenizeString2( pszCommand, " \t\n",
                        CSLT_HONOURSTRINGS | CSLT_ALLOWEMPTYTOKENS );

for( i = 0; papszTokens != NULL && papszTokens[i] != NULL; i++ )
    printf( "arg %d: '%s'", papszTokens[i] );
CSLDestroy( papszTokens );
```

Parameters:

pszString the string to be split into tokens.

pszDelimiters one or more characters to be used as token delimeters.

nCSLTFlags an ORing of one or more of the CSLT_ flag values.

Returns:

a string list of tokens owned by the caller.

44.10 cpl_vsi.h File Reference

```
#include "cpl_port.h"
#include <unistd.h>
#include <sys/stat.h>
```

Defines

- #define **VSI_ISLNK**(x) S_ISLNK(x)
- #define **VSI_ISREG**(x) S_ISREG(x)
- #define **VSI_ISDIR**(x) S_ISDIR(x)
- #define **VSI_ISCHR**(x) S_ISCHR(x)
- #define **VSI_ISBLK**(x) S_ISBLK(x)
- #define **CPLReadDir** VSIReadDir
- #define **VSIDebug4**(f, a1, a2, a3, a4) {}
- #define **VSIDebug3**(f, a1, a2, a3) {}
- #define **VSIDebug2**(f, a1, a2) {}
- #define **VSIDebug1**(f, a1) {}

Typedefs

- typedef struct stat **VSIStatBuf**
- typedef GUIntBig **vsi_l_offset**
- typedef struct VSI_STAT64_T **VSIStatBufL**

Functions

- FILE * **VSIFOpen** (const char *, const char *)
- int **VSIFClose** (FILE *)
- int **VSIFSeek** (FILE *, long, int)
- long **VSIFTell** (FILE *)
- void **VSIRewind** (FILE *)
- void **VSIFFlush** (FILE *)
- size_t **VSIFRead** (void *, size_t, size_t, FILE *)
- size_t **VSIFWrite** (const void *, size_t, size_t, FILE *)
- char * **VSIFGets** (char *, int, FILE *)
- int **VSIFPuts** (const char *, FILE *)
- int **VSIFPrintf** (FILE *, const char *,...) CPL_PRINT_FUNC_FORMAT(2
- int int **VSIFGetc** (FILE *)
- int **VSIFPutc** (int, FILE *)
- int **VSIUngetc** (int, FILE *)
- int **VSIFEof** (FILE *)
- int **VSIStat** (const char *, VSIStatBuf *)
- FILE * **VSIFOpenL** (const char *, const char *)

  Open file.

- int **VSIFCloseL** (FILE *)

  Close file.

- int **VSIFSeekL** (FILE *, vsi_l_offset, int)

  Seek to requested offset.

- vsi_l_offset **VSIFTellL** (FILE *)

  Tell current file offset.

- void **VSIRewindL** (FILE *)
- size_t **VSIFReadL** (void *, size_t, size_t, FILE *)

  Read bytes from file.

- size_t **VSIFWriteL** (const void *, size_t, size_t, FILE *)

  Write bytes to file.

- int **VSIFEofL** (FILE *)

  Test for end of file.

- int **VSIFFlushL** (FILE *)

  Flush pending writes to disk.

- int **VSIFPrintfL** (FILE *, const char *,...) CPL_PRINT_FUNC_FORMAT(2
- int int **VSIFPutcL** (int, FILE *)
- int **VSIStatL** (const char *, VSIStatBufL *)

  Get filesystem object info.

- void * **VSICalloc** (size_t, size_t)
- void * **VSIMalloc** (size_t)
- void **VSIFree** (void *)
- void * **VSIRealloc** (void *, size_t)
- char * **VSIStrdup** (const char *)
- void * **VSIMalloc2** (size_t nSize1, size_t nSize2)
- void * **VSIMalloc3** (size_t nSize1, size_t nSize2, size_t nSize3)
- char ** **VSIReadDir** (const char *)

  Read names in a directory.

- int **VSIMkdir** (const char *pathname, long mode)

  Create a directory.

- int **VSIRmdir** (const char *pathname)

  Delete a directory.

- int **VSIUnlink** (const char *pathname)

  Delete a file.

- int **VSIRename** (const char *oldpath, const char *newpath)

  Rename a file.

- char * **VSIStrerror** (int)
- void **VSIInstallMemFileHandler** (void)

  Install "memory" file system handler.

- void **VSIInstallLargeFileHandler** (void)
- void **VSIInstallGZipFileHandler** (void)

  Install GZip file system handler.

- void **VSIInstallZipFileHandler** (void)

  Install ZIP file system handler.

- void **VSICleanupFileManager** (void)
- FILE * **VSIFileFromMemBuffer** (const char *pszFilename, GByte *pabyData, vsi_l_offset nDataLength, int bTakeOwnership)

  Create memory "file" from a buffer.

- GByte * **VSIGetMemFileBuffer** (const char *pszFilename, vsi_l_offset *pnDataLength, int bUnlinkAndSeize)

  Fetch buffer underlying memory file.

- unsigned long **VSITime** (unsigned long *)
- const char * **VSICTime** (unsigned long)
- struct tm * **VSIGMTime** (const time_t *pnTime, struct tm *poBrokenTime)
- struct tm * **VSILocalTime** (const time_t *pnTime, struct tm *poBrokenTime)

44.10.1 Detailed Description

Standard C Covers

The VSI functions are intended to be hookable aliases for Standard C I/O, memory allocation and other system functions. They are intended to allow virtualization of disk I/O so that non file data sources can be made to appear as files, and so that additional error trapping and reporting can be interested. The memory access API is aliased so that special application memory management services can be used.

Is is intended that each of these functions retains exactly the same calling pattern as the original Standard C functions they relate to. This means we don't have to provide custom documentation, and also means that the default implementation is very simple.

44.10.2 Function Documentation

44.10.2.1 int VSIFCloseL (FILE * *fp*)

Close file.

This function closes the indicated file.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fclose() function.

Parameters:

fp file handle opened with **VSIFOpenL()** (p. **??**).

Returns:

0 on success or -1 on failure.

References VSIFCloseL().

Referenced by CPLCloseShared(), CPLParseXMLFile(), CPLSerializeXMLTreeToFile(), CSLLoad(), GDALVersionInfo(), GDALWriteWorldFile(), and VSIFCloseL().

44.10.2.2 int VSIFEofL (FILE * *fp*)

Test for end of file.

Returns TRUE (non-zero) if the file read/write offset is currently at the end of the file.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX feof() call.

Parameters:

fp file handle opened with **VSIFOpenL()** (p. **??**).

Returns:

TRUE if at EOF else FALSE.

References VSIFEofL().

Referenced by CSLLoad(), and VSIFEofL().

44.10.2.3 int VSIFFlushL (FILE * *fp*)

Flush pending writes to disk.

For files in write or update mode and on filesystem types where it is applicable, all pending output on the file is flushed to the physical disk.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fflush() call.

Parameters:

fp file handle opened with **VSIFOpenL()** (p. **??**).

Returns:

0 on success or -1 on error.

References VSIFFlushL().

Referenced by VSIFFlushL().

44.10.2.4 FILE* VSIFileFromMemBuffer (const char * *pszFilename*, GByte * *pabyData*, vsi_l_offset *nDataLength*, int *bTakeOwnership*)

Create memory "file" from a buffer.

A virtual memory file is created from the passed buffer with the indicated filename. Under normal conditions the filename would need to be absolute and within the /vsimem/ portion of the filesystem.

If bTakeOwnership is TRUE, then the memory file system handler will take ownership of the buffer, freeing it when the file is deleted. Otherwise it remains the responsibility of the caller, but should not be freed as long as it might be accessed as a file. In no circumstances does this function take a copy of the pabyData contents.

Parameters:

pszFilename the filename to be created.

pabyData the data buffer for the file.

nDataLength the length of buffer in bytes.

bTakeOwnership TRUE to transfer "ownership" of buffer or FALSE.

Returns:

open file handle on created file (see **VSIFOpenL()** (p. **??**)).

References VSIFileFromMemBuffer(), and VSIInstallMemFileHandler().

Referenced by VSIFileFromMemBuffer().

44.10.2.5 FILE* VSIFOpenL (const char * *pszFilename*, const char * *pszAccess*)

Open file.

This function opens a file with the desired access. Large files (larger than 2GB) should be supported. Binary access is always implied and the "b" does not need to be included in the pszAccess string.

Note that the "FILE *" returned by this function is not really a standard C library FILE *, and cannot be used with any functions other than the "VSI*L" family of functions. They aren't "real" FILE objects.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fopen() function.

Parameters:

pszFilename the file to open.

pszAccess access requested (ie. "r", "r+", "w".

Returns:

NULL on failure, or the file handle.

References VSIFOpenL().

Referenced by CPLOpenShared(), CPLParseXMLFile(), CPLSerializeXMLTreeToFile(), CSLLoad(), GDALVersionInfo(), GDALWriteWorldFile(), and VSIFOpenL().

44.10.2.6 size_t VSIFReadL (void * *pBuffer*, size_t *nSize*, size_t *nCount*, FILE * *fp*)

Read bytes from file.

Reads nCount objects of nSize bytes from the indicated file at the current offset into the indicated buffer.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fread() call.

Parameters:

pBuffer the buffer into which the data should be read (at least nCount ∗ nSize bytes in size.

nSize size of objects to read in bytes.

nCount number of objects to read.

fp file handle opened with **VSIFOpenL()** (p. **??**).

Returns:

number of objects successfully read.

References VSIFReadL().

Referenced by CPLParseXMLFile(), CPLReadLineL(), GDALVersionInfo(), and VSIFReadL().

44.10.2.7 int VSIFSeekL (FILE ∗ *fp*, vsi_l_offset *nOffset*, int *nWhence*)

Seek to requested offset.

Seek to the desired offset (nOffset) in the indicated file.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fseek() call.

Parameters:

fp file handle opened with **VSIFOpenL()** (p. **??**).

nOffset offset in bytes.

nWhence one of SEEK_SET, SEEK_CUR or SEEK_END.

Returns:

0 on success or -1 one failure.

References VSIFSeekL().

Referenced by CPLParseXMLFile(), CPLReadLineL(), GDALVersionInfo(), and VSIFSeekL().

44.10.2.8 vsi_l_offset VSIFTellL (FILE ∗ *fp*)

Tell current file offset.

Returns the current file read/write offset in bytes from the beginning of the file.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX ftell() call.

Parameters:

fp file handle opened with **VSIFOpenL()** (p. **??**).

Returns:

file offset in bytes.

References VSIFTellL().

Referenced by CPLParseXMLFile(), CPLReadLineL(), GDALVersionInfo(), and VSIFTellL().

44.10.2.9 size_t VSIFWriteL (const void * *pBuffer*, size_t *nSize*, size_t *nCount*, FILE * *fp*)

Write bytes to file.

Writess nCount objects of nSize bytes to the indicated file at the current offset into the indicated buffer.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fwrite() call.

Parameters:

pBuffer the buffer from which the data should be written (at least nCount * nSize bytes in size.

nSize size of objects to read in bytes.

nCount number of objects to read.

fp file handle opened with **VSIFOpenL()** (p. **??**).

Returns:

number of objects successfully written.

References VSIFWriteL().

Referenced by CPLSerializeXMLTreeToFile(), GDALWriteWorldFile(), and VSIFWriteL().

44.10.2.10 GByte* VSIGetMemFileBuffer (const char * *pszFilename*, vsi_l_offset * *pnDataLength*, int *bUnlinkAndSeize*)

Fetch buffer underlying memory file.

This function returns a pointer to the memory buffer underlying a virtual "in memory" file. If bUnlinkAndSeize is TRUE the filesystem object will be deleted, and ownership of the buffer will pass to the caller otherwise the underlying file will remain in existance.

Parameters:

pszFilename the name of the file to grab the buffer of.

pnDataLength (file) length returned in this variable.

bUnlinkAndSeize TRUE to remove the file, or FALSE to leave unaltered.

Returns:

pointer to memory buffer or NULL on failure.

References VSIGetMemFileBuffer().

Referenced by VSIGetMemFileBuffer().

44.10.2.11 void VSIInstallGZipFileHandler (void)

Install GZip file system handler.

A special file handler is installed that allows reading on-the-fly in GZip (.gz) files. All portions of the file system underneath the base path "/vsigzip/" will be handled by this driver.

Additional documentation is to be found at `http://trac.osgeo.org/gdal/wiki/UserDocs/ReadInZip`

References VSIInstallGZipFileHandler().

Referenced by VSIInstallGZipFileHandler().

44.10.2.12 void VSIInstallMemFileHandler (void)

Install "memory" file system handler.

A special file handler is installed that allows block of memory to be treated as files. All portions of the file system underneath the base path "/vsimem/" will be handled by this driver.

Normal VSI∗L functions can be used freely to create and destroy memory arrays treating them as if they were real file system objects. Some additional methods exist to efficient create memory file system objects without duplicating original copies of the data or to "steal" the block of memory associated with a memory file.

At this time the memory handler does not properly handle directory semantics for the memory portion of the filesystem. The **VSIReadDir()** (p. **??**) function is not supported though this will be corrected in the future.

Calling this function repeatedly should do no harm, though it is not necessary. It is already called the first time a virtualizable file access function (ie. **VSIFOpenL()** (p. **??**), VSIMkDir(), etc) is called.

This code example demonstrates using GDAL to translate from one memory buffer to another.

```
GByte *ConvertBufferFormat( GByte *pabyInData, vsi_l_offset nInDataLength,
                            vsi_l_offset *pnOutDataLength )
{
    // create memory file system object from buffer.
    VSIFCloseL( VSIFileFromMemBuffer( "/vsimem/work.dat", pabyInData,
                                      nInDataLength, FALSE ) );

    // Open memory buffer for read.
    GDALDatasetH hDS = GDALOpen( "/vsimem/work.dat", GA_ReadOnly );

    // Get output format driver.
    GDALDriverH hDriver = GDALGetDriverByName( "GTiff" );
    GDALDatasetH hOutDS;

    hOutDS = GDALCreateCopy( hDriver, "/vsimem/out.tif", hDS, TRUE, NULL,
                             NULL, NULL );

    // close source file, and "unlink" it.
    GDALClose( hDS );
    VSIUnlink( "/vsimem/work.dat" );

    // seize the buffer associated with the output file.

    return VSIGetMemFileBuffer( "/vsimem/out.tif", pnOutDataLength, TRUE );
}
```

References VSIInstallMemFileHandler().

Referenced by VSIFileFromMemBuffer(), and VSIInstallMemFileHandler().

44.10.2.13 void VSIInstallZipFileHandler (void)

Install ZIP file system handler.

A special file handler is installed that allows reading on-the-fly in ZIP (.zip) archives. All portions of the file system underneath the base path "/vsizip/" will be handled by this driver.

Additional documentation is to be found at `http://trac.osgeo.org/gdal/wiki/UserDocs/ReadInZip`

References VSIInstallZipFileHandler().

Referenced by VSIInstallZipFileHandler().

44.10.2.14 void* VSIMalloc2 (size_t *nSize1*, size_t *nSize2*)

VSIMalloc2 allocates (nSize1 * nSize2) bytes. In case of overflow of the multiplication, or if memory allocation fails, a NULL pointer is returned and a CE_Failure error is raised with CPLError(). If nSize1 == 0 || nSize2 == 0, a NULL pointer will also be returned. CPLFree() or VSIFree() can be used to free memory allocated by this function.

References VSIMalloc2().

Referenced by GDALChecksumImage(), GDALRegenerateOverviews(), and VSIMalloc2().

44.10.2.15 void* VSIMalloc3 (size_t *nSize1*, size_t *nSize2*, size_t *nSize3*)

VSIMalloc3 allocates (nSize1 * nSize2 * nSize3) bytes. In case of overflow of the multiplication, or if memory allocation fails, a NULL pointer is returned and a CE_Failure error is raised with CPLError(). If nSize1 == 0 || nSize2 == 0 || nSize3 == 0, a NULL pointer will also be returned. CPLFree() or VSIFree() can be used to free memory allocated by this function.

References VSIMalloc3().

Referenced by GDALDatasetCopyWholeRaster(), GDALRegenerateOverviews(), and VSIMalloc3().

44.10.2.16 int VSIMkdir (const char * *pszPathname*, long *mode*)

Create a directory.

Create a new directory with the indicated mode. The mode is ignored on some platforms. A reasonable default mode value would be 0666. This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX mkdir() function.

Parameters:

pszPathname the path to the directory to create.

mode the permissions mode.

Returns:

0 on success or -1 on an error.

References VSIMkdir().

Referenced by VSIMkdir().

44.10.2.17 char** VSIReadDir (const char * *pszPath*)

Read names in a directory.

This function abstracts access to directory contains. It returns a list of strings containing the names of files, and directories in this directory. The resulting string list becomes the responsibility of the application and should be freed with **CSLDestroy()** (p. **??**) when no longer needed.

Note that no error is issued via CPLError() if the directory path is invalid, though NULL is returned.

This function used to be known as CPLReadDir(), but the old name is now deprecated.

Parameters:

pszPath the relative, or absolute path of a directory to read.

Returns:

The list of entries in the directory, or NULL if the directory doesn't exist.

References VSIReadDir().

Referenced by VSIReadDir().

44.10.2.18 int VSIRename (const char * *oldpath*, const char * *newpath*)

Rename a file.

Renames a file object in the file system. It should be possible to rename a file onto a new filesystem, but it is safest if this function is only used to rename files that remain in the same directory.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX rename() function.

Parameters:

oldpath the name of the file to be renamed.

newpath the name the file should be given.

Returns:

0 on success or -1 on an error.

References VSIRename().

Referenced by VSIRename().

44.10.2.19 int VSIRmdir (const char * *pszDirname*)

Delete a directory.

Deletes a directory object from the file system. On some systems the directory must be empty before it can be deleted.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX rmdir() function.

Parameters:

pszDirname the path of the directory to be deleted.

Returns:

0 on success or -1 on an error.

References VSIRmdir().

Referenced by CPLUnlinkTree(), and VSIRmdir().

44.10.2.20 int VSIStatL (const char * *pszFilename*, VSIStatBufL * *psStatBuf*)

Get filesystem object info.

Fetches status information about a filesystem object (file, directory, etc). The returned information is placed in the VSIStatBufL structure. For portability only the st_size (size in bytes), and st_mode (file type). This method is similar to VSIStat(), but will work on large files on systems where this requires special calls.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX stat() function.

Parameters:

pszFilename the path of the filesystem object to be queried.

psStatBuf the structure to load with information.

Returns:

0 on success or -1 on an error.

References VSIStatL().

Referenced by CPLCheckForFile(), CPLFormCIFilename(), GDALReadWorldFile(), GDALPamDataset::GetFileList(), GDALDataset::GetFileList(), and VSIStatL().

44.10.2.21 int VSIUnlink (const char * *pszFilename*)

Delete a file.

Deletes a file object from the file system.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX unlink() function.

Parameters:

pszFilename the path of the file to be deleted.

Returns:

0 on success or -1 on an error.

References VSIUnlink().

Referenced by GDALDriver::CopyFiles(), CPLUnlinkTree(), GDALDriver::Delete(), and VSIUnlink().

44.11 gdal.h File Reference

```
#include "gdal_version.h"
#include "cpl_port.h"
#include "cpl_error.h"
```

Classes

- struct **GDALOptionDefinition**
- struct **GDAL_GCP**
- struct **GDALRPCInfo**
- struct **GDALColorEntry**

Defines

- #define **GDALMD_AREA_OR_POINT** "AREA_OR_POINT"
- #define **GDALMD_AOP_AREA** "Area"
- #define **GDALMD_AOP_POINT** "Point"
- #define **CPLE_WrongFormat** 200
- #define **GDAL_DMD_LONGNAME** "DMD_LONGNAME"
- #define **GDAL_DMD_HELPTOPIC** "DMD_HELPTOPIC"
- #define **GDAL_DMD_MIMETYPE** "DMD_MIMETYPE"
- #define **GDAL_DMD_EXTENSION** "DMD_EXTENSION"
- #define **GDAL_DMD_CREATIONOPTIONLIST** "DMD_CREATIONOPTIONLIST"
- #define **GDAL_DMD_CREATIONDATATYPES** "DMD_CREATIONDATATYPES"
- #define **GDAL_DCAP_CREATE** "DCAP_CREATE"
- #define **GDAL_DCAP_CREATECOPY** "DCAP_CREATECOPY"
- #define **GDAL_DCAP_VIRTUALIO** "DCAP_VIRTUALIO"
- #define **SRCVAL**(papoSource, eSrcType, ii)
- #define **GMF_ALL_VALID** 0x01
- #define **GMF_PER_DATASET** 0x02
- #define **GMF_ALPHA** 0x04
- #define **GMF_NODATA** 0x08
- #define **GDAL_CHECK_VERSION**(pszCallingComponentName) GDALCheckVersion(GDAL_VERSION_MAJOR, GDAL_VERSION_MINOR, pszCallingComponentName)

Typedefs

- typedef void * **GDALMajorObjectH**
- typedef void * **GDALDatasetH**
- typedef void * **GDALRasterBandH**
- typedef void * **GDALDriverH**
- typedef void * **GDALProjDefH**
- typedef void * **GDALColorTableH**
- typedef void * **GDALRasterAttributeTableH**
- typedef const char void *int **GDALDummyProgress** (double, const char *, void *)
- typedef CPLErr(* **GDALDerivedPixelFunc**)(void **papoSources, int nSources, void *pData, int nBufXSize, int nBufYSize, **GDALDataType** eSrcType, **GDALDataType** eBufType, int nPixelSpace, int nLineSpace)

Enumerations

- enum **GDALDataType** {
  GDT_Unknown = 0, **GDT_Byte** = 1, **GDT_UInt16** = 2, **GDT_Int16** = 3,
  GDT_UInt32 = 4, **GDT_Int32** = 5, **GDT_Float32** = 6, **GDT_Float64** = 7,
  GDT_CInt16 = 8, **GDT_CInt32** = 9, **GDT_CFloat32** = 10, **GDT_CFloat64** = 11,
  GDT_TypeCount = 12 }
- enum **GDALAccess** { **GA_ReadOnly** = 0, **GA_Update** = 1 }
- enum **GDALRWFlag** { **GF_Read** = 0, **GF_Write** = 1 }
- enum **GDALColorInterp** {
  GCI_Undefined = 0, **GCI_GrayIndex** = 1, **GCI_PaletteIndex** = 2, **GCI_RedBand** = 3,
  GCI_GreenBand = 4, **GCI_BlueBand** = 5, **GCI_AlphaBand** = 6, **GCI_HueBand** = 7,
  GCI_SaturationBand = 8, **GCI_LightnessBand** = 9, **GCI_CyanBand** = 10, **GCI_MagentaBand** = 11,
  GCI_YellowBand = 12, **GCI_BlackBand** = 13, **GCI_YCbCr_YBand** = 14, **GCI_YCbCr_CbBand** = 15,
  GCI_YCbCr_CrBand = 16, **GCI_Max** = 16 }
- enum **GDALPaletteInterp** { **GPI_Gray** = 0, **GPI_RGB** = 1, **GPI_CMYK** = 2, **GPI_HLS** = 3 }
- enum **GDALRATFieldType** { **GFT_Integer**, **GFT_Real**, **GFT_String** }
- enum **GDALRATFieldUsage** {
  GFU_Generic = 0, **GFU_PixelCount** = 1, **GFU_Name** = 2, **GFU_Min** = 3,
  GFU_Max = 4, **GFU_MinMax** = 5, **GFU_Red** = 6, **GFU_Green** = 7,
  GFU_Blue = 8, **GFU_Alpha** = 9, **GFU_RedMin** = 10, **GFU_GreenMin** = 11,
  GFU_BlueMin = 12, **GFU_AlphaMin** = 13, **GFU_RedMax** = 14, **GFU_GreenMax** = 15,
  GFU_BlueMax = 16, **GFU_AlphaMax** = 17, **GFU_MaxCount** }

Functions

- int **GDALGetDataTypeSize** (**GDALDataType**)
- int **GDALDataTypeIsComplex** (**GDALDataType**)
- const char * **GDALGetDataTypeName** (**GDALDataType**)
- **GDALDataType GDALGetDataTypeByName** (const char *)
- **GDALDataType GDALDataTypeUnion** (**GDALDataType**, **GDALDataType**)
- const char * **GDALGetColorInterpretationName** (**GDALColorInterp**)
- const char * **GDALGetPaletteInterpretationName** (**GDALPaletteInterp**)
- typedef **int** (1 *GDALProgressFunc)(double
- int **GDALTermProgress** (double, const char *, void *)
- int **GDALScaledProgress** (double, const char *, void *)
- void * **GDALCreateScaledProgress** (double, double, GDALProgressFunc, void *)
- void **GDALDestroyScaledProgress** (void *)
- void **GDALAllRegister** (void)
- GDALDatasetH **GDALCreate** (GDALDriverH hDriver, const char *, int, int, int, **GDALDataType**, char **)
- GDALDatasetH **GDALCreateCopy** (GDALDriverH, const char *, GDALDatasetH, int, char **, GDALProgressFunc, void *)
- GDALDriverH **GDALIdentifyDriver** (const char *pszFilename, char **papszFileList)
- GDALDatasetH **GDALOpen** (const char *pszFilename, **GDALAccess** eAccess)

- GDALDatasetH **GDALOpenShared** (const char *, **GDALAccess**)
- int **GDALDumpOpenDatasets** (FILE *)
- GDALDriverH **GDALGetDriverByName** (const char *)
- int **GDALGetDriverCount** (void)
- GDALDriverH **GDALGetDriver** (int)
- void **GDALDestroyDriver** (GDALDriverH)
- int **GDALRegisterDriver** (GDALDriverH)
- void **GDALDeregisterDriver** (GDALDriverH)
- void **GDALDestroyDriverManager** (void)
- CPLErr **GDALDeleteDataset** (GDALDriverH, const char *)
- CPLErr **GDALRenameDataset** (GDALDriverH, const char *pszNewName, const char *pszOldName)
- CPLErr **GDALCopyDatasetFiles** (GDALDriverH, const char *pszNewName, const char *pszOldName)
- int **GDALValidateCreationOptions** (GDALDriverH, char **papszCreationOptions)
- const char * **GDALGetDriverShortName** (GDALDriverH)
- const char * **GDALGetDriverLongName** (GDALDriverH)
- const char * **GDALGetDriverHelpTopic** (GDALDriverH)
- const char * **GDALGetDriverCreationOptionList** (GDALDriverH)
- void **GDALInitGCPs** (int, **GDAL_GCP** *)
- void **GDALDeinitGCPs** (int, **GDAL_GCP** *)
- **GDAL_GCP** * **GDALDuplicateGCPs** (int, const **GDAL_GCP** *)
- int **GDALGCPsToGeoTransform** (int nGCPCount, const **GDAL_GCP** *pasGCPs, double *padfGeoTransform, int bApproxOK)
- int **GDALInvGeoTransform** (double *padfGeoTransformIn, double *padfInvGeoTransformOut)
- void **GDALApplyGeoTransform** (double *, double, double, double *, double *)
- char ** **GDALGetMetadata** (GDALMajorObjectH, const char *)
- CPLErr **GDALSetMetadata** (GDALMajorObjectH, char **, const char *)
- const char * **GDALGetMetadataItem** (GDALMajorObjectH, const char *, const char *)
- CPLErr **GDALSetMetadataItem** (GDALMajorObjectH, const char *, const char *, const char *)
- const char * **GDALGetDescription** (GDALMajorObjectH)
- void **GDALSetDescription** (GDALMajorObjectH, const char *)
- GDALDriverH **GDALGetDatasetDriver** (GDALDatasetH)
- char ** **GDALGetFileList** (GDALDatasetH)
- void **GDALClose** (GDALDatasetH)
- int **GDALGetRasterXSize** (GDALDatasetH)
- int **GDALGetRasterYSize** (GDALDatasetH)
- int **GDALGetRasterCount** (GDALDatasetH)
- GDALRasterBandH **GDALGetRasterBand** (GDALDatasetH, int)
- CPLErr **GDALAddBand** (GDALDatasetH hDS, **GDALDataType** eType, char **papszOptions)
- CPLErr **GDALDatasetRasterIO** (GDALDatasetH hDS, **GDALRWFlag** eRWFlag, int nDSXOff, int nDSYOff, int nDSXSize, int nDSYSize, void *pBuffer, int nBXSize, int nBYSize, **GDALDataType** eBDataType, int nBandCount, int *panBandCount, int nPixelSpace, int nLineSpace, int nBandSpace)
- CPLErr **GDALDatasetAdviseRead** (GDALDatasetH hDS, int nDSXOff, int nDSYOff, int nDSXSize, int nDSYSize, int nBXSize, int nBYSize, **GDALDataType** eBDataType, int nBandCount, int *panBandCount, char **papszOptions)
- const char * **GDALGetProjectionRef** (GDALDatasetH)
- CPLErr **GDALSetProjection** (GDALDatasetH, const char *)
- CPLErr **GDALGetGeoTransform** (GDALDatasetH, double *)
- CPLErr **GDALSetGeoTransform** (GDALDatasetH, double *)

- int **GDALGetGCPCount** (GDALDatasetH)
- const char * **GDALGetGCPProjection** (GDALDatasetH)
- const **GDAL_GCP** * **GDALGetGCPs** (GDALDatasetH)
- CPLErr **GDALSetGCPs** (GDALDatasetH, int, const **GDAL_GCP** *, const char *)
- void * **GDALGetInternalHandle** (GDALDatasetH, const char *)
- int **GDALReferenceDataset** (GDALDatasetH)
- int **GDALDereferenceDataset** (GDALDatasetH)
- CPLErr **GDALBuildOverviews** (GDALDatasetH, const char *, int, int *, int, int *, GDALProgressFunc, void *)
- void **GDALGetOpenDatasets** (GDALDatasetH **hDS, int *pnCount)
- int **GDALGetAccess** (GDALDatasetH hDS)
- void **GDALFlushCache** (GDALDatasetH hDS)
- CPLErr **GDALCreateDatasetMaskBand** (GDALDatasetH hDS, int nFlags)
- CPLErr **GDALDatasetCopyWholeRaster** (GDALDatasetH hSrcDS, GDALDatasetH hDstDS, char **papszOptions, GDALProgressFunc pfnProgress, void *pProgressData)
- CPLErr **GDALRegenerateOverviews** (GDALRasterBandH hSrcBand, int nOverviewCount, GDALRasterBandH *pahOverviewBands, const char *pszResampling, GDALProgressFunc pfnProgress, void *pProgressData)
- **GDALDataType GDALGetRasterDataType** (GDALRasterBandH)
- void **GDALGetBlockSize** (GDALRasterBandH, int *pnXSize, int *pnYSize)
- CPLErr **GDALRasterAdviseRead** (GDALRasterBandH hRB, int nDSXOff, int nDSYOff, int nDSXSize, int nDSYSize, int nBXSize, int nBYSize, **GDALDataType** eBDataType, char **papszOptions)
- CPLErr **GDALRasterIO** (GDALRasterBandH hRBand, **GDALRWFlag** eRWFlag, int nDSXOff, int nDSYOff, int nDSXSize, int nDSYSize, void *pBuffer, int nBXSize, int nBYSize, **GDALDataType** eBDataType, int nPixelSpace, int nLineSpace)
- CPLErr **GDALReadBlock** (GDALRasterBandH, int, int, void *)
- CPLErr **GDALWriteBlock** (GDALRasterBandH, int, int, void *)
- int **GDALGetRasterBandXSize** (GDALRasterBandH)
- int **GDALGetRasterBandYSize** (GDALRasterBandH)
- **GDALAccess GDALGetRasterAccess** (GDALRasterBandH)
- int **GDALGetBandNumber** (GDALRasterBandH)
- GDALDatasetH **GDALGetBandDataset** (GDALRasterBandH)
- **GDALColorInterp GDALGetRasterColorInterpretation** (GDALRasterBandH)
- CPLErr **GDALSetRasterColorInterpretation** (GDALRasterBandH, **GDALColorInterp**)
- GDALColorTableH **GDALGetRasterColorTable** (GDALRasterBandH)
- CPLErr **GDALSetRasterColorTable** (GDALRasterBandH, GDALColorTableH)
- int **GDALHasArbitraryOverviews** (GDALRasterBandH)
- int **GDALGetOverviewCount** (GDALRasterBandH)
- GDALRasterBandH **GDALGetOverview** (GDALRasterBandH, int)
- double **GDALGetRasterNoDataValue** (GDALRasterBandH, int *)
- CPLErr **GDALSetRasterNoDataValue** (GDALRasterBandH, double)
- char ** **GDALGetRasterCategoryNames** (GDALRasterBandH)
- CPLErr **GDALSetRasterCategoryNames** (GDALRasterBandH, char **)
- double **GDALGetRasterMinimum** (GDALRasterBandH, int *pbSuccess)
- double **GDALGetRasterMaximum** (GDALRasterBandH, int *pbSuccess)
- CPLErr **GDALGetRasterStatistics** (GDALRasterBandH, int bApproxOK, int bForce, double *pdfMin, double *pdfMax, double *pdfMean, double *pdfStdDev)
- CPLErr **GDALComputeRasterStatistics** (GDALRasterBandH, int bApproxOK, double *pdfMin, double *pdfMax, double *pdfMean, double *pdfStdDev, GDALProgressFunc pfnProgress, void *pProgressData)

- CPLErr **GDALSetRasterStatistics** (GDALRasterBandH hBand, double dfMin, double dfMax, double dfMean, double dfStdDev)
- const char * **GDALGetRasterUnitType** (GDALRasterBandH)
- double **GDALGetRasterOffset** (GDALRasterBandH, int *pbSuccess)
- CPLErr **GDALSetRasterOffset** (GDALRasterBandH hBand, double dfNewOffset)
- double **GDALGetRasterScale** (GDALRasterBandH, int *pbSuccess)
- CPLErr **GDALSetRasterScale** (GDALRasterBandH hBand, double dfNewOffset)
- void **GDALComputeRasterMinMax** (GDALRasterBandH hBand, int bApproxOK, double adfMinMax[2])
- CPLErr **GDALFlushRasterCache** (GDALRasterBandH hBand)
- CPLErr **GDALGetRasterHistogram** (GDALRasterBandH hBand, double dfMin, double dfMax, int nBuckets, int *panHistogram, int bIncludeOutOfRange, int bApproxOK, GDALProgressFunc pfnProgress, void *pProgressData)
- CPLErr **GDALGetDefaultHistogram** (GDALRasterBandH hBand, double *pdfMin, double *pdfMax, int *pnBuckets, int **ppanHistogram, int bForce, GDALProgressFunc pfnProgress, void *pProgressData)
- CPLErr **GDALSetDefaultHistogram** (GDALRasterBandH hBand, double dfMin, double dfMax, int nBuckets, int *panHistogram)
- int **GDALGetRandomRasterSample** (GDALRasterBandH, int, float *)
- GDALRasterBandH **GDALGetRasterSampleOverview** (GDALRasterBandH, int)
- CPLErr **GDALFillRaster** (GDALRasterBandH hBand, double dfRealValue, double dfImaginaryValue)
- CPLErr **GDALComputeBandStats** (GDALRasterBandH hBand, int nSampleStep, double *pdfMean, double *pdfStdDev, GDALProgressFunc pfnProgress, void *pProgressData)
- CPLErr **GDALOverviewMagnitudeCorrection** (GDALRasterBandH hBaseBand, int nOverviewCount, GDALRasterBandH *pahOverviews, GDALProgressFunc pfnProgress, void *pProgressData)
- GDALRasterAttributeTableH **GDALGetDefaultRAT** (GDALRasterBandH hBand)
- CPLErr **GDALSetDefaultRAT** (GDALRasterBandH, GDALRasterAttributeTableH)
- CPLErr **GDALAddDerivedBandPixelFunc** (const char *pszName, GDALDerivedPixelFunc pfnPixelFunc)
- GDALRasterBandH **GDALGetMaskBand** (GDALRasterBandH hBand)
- int **GDALGetMaskFlags** (GDALRasterBandH hBand)
- CPLErr **GDALCreateMaskBand** (GDALRasterBandH hBand, int nFlags)
- int **GDALGeneralCmdLineProcessor** (int nArgc, char ***ppapszArgv, int nOptions)
- void **GDALSwapWords** (void *pData, int nWordSize, int nWordCount, int nWordSkip)
- void **GDALCopyWords** (void *pSrcData, **GDALDataType** eSrcType, int nSrcPixelOffset, void *pDstData, **GDALDataType** eDstType, int nDstPixelOffset, int nWordCount)
- void **GDALCopyBits** (const GByte *pabySrcData, int nSrcOffset, int nSrcStep, GByte *pabyDstData, int nDstOffset, int nDstStep, int nBitCount, int nStepCount)
- int **GDALLoadWorldFile** (const char *, double *)
- int **GDALReadWorldFile** (const char *, const char *, double *)
- int **GDALWriteWorldFile** (const char *, const char *, double *)
- int **GDALLoadTabFile** (const char *, double *, char **, int *, **GDAL_GCP** **)
- int **GDALReadTabFile** (const char *, double *, char **, int *, **GDAL_GCP** **)
- char ** **GDALLoadRPBFile** (const char *pszFilename, char **papszSiblingFiles)
- CPLErr **GDALWriteRPBFile** (const char *pszFilename, char **papszMD)
- char ** **GDALLoadIMDFile** (const char *pszFilename, char **papszSiblingFiles)
- CPLErr **GDALWriteIMDFile** (const char *pszFilename, char **papszMD)
- const char * **GDALDecToDMS** (double, const char *, int)
- double **GDALPackedDMSToDec** (double)

- double **GDALDecToPackedDMS** (double)
- const char * **GDALVersionInfo** (const char *)
- int **GDALCheckVersion** (int nVersionMajor, int nVersionMinor, const char *pszCallingComponentName)
- int **GDALExtractRPCInfo** (char **, GDALRPCInfo *)
- GDALColorTableH **GDALCreateColorTable** (**GDALPaletteInterp**)
- void **GDALDestroyColorTable** (GDALColorTableH)
- GDALColorTableH **GDALCloneColorTable** (GDALColorTableH)
- **GDALPaletteInterp GDALGetPaletteInterpretation** (GDALColorTableH)
- int **GDALGetColorEntryCount** (GDALColorTableH)
- const **GDALColorEntry** * **GDALGetColorEntry** (GDALColorTableH, int)
- int **GDALGetColorEntryAsRGB** (GDALColorTableH, int, **GDALColorEntry** *)
- void **GDALSetColorEntry** (GDALColorTableH, int, const **GDALColorEntry** *)
- void **GDALCreateColorRamp** (GDALColorTableH hTable, int nStartIndex, const **GDALColorEntry** *psStartColor, int nEndIndex, const **GDALColorEntry** *psEndColor)
- GDALRasterAttributeTableH **GDALCreateRasterAttributeTable** (void)
- void **GDALDestroyRasterAttributeTable** (GDALRasterAttributeTableH)
- int **GDALRATGetColumnCount** (GDALRasterAttributeTableH)
- const char * **GDALRATGetNameOfCol** (GDALRasterAttributeTableH, int)
- **GDALRATFieldUsage GDALRATGetUsageOfCol** (GDALRasterAttributeTableH, int)
- **GDALRATFieldType GDALRATGetTypeOfCol** (GDALRasterAttributeTableH, int)
- int **GDALRATGetColOfUsage** (GDALRasterAttributeTableH, **GDALRATFieldUsage**)
- int **GDALRATGetRowCount** (GDALRasterAttributeTableH)
- const char * **GDALRATGetValueAsString** (GDALRasterAttributeTableH, int, int)
- int **GDALRATGetValueAsInt** (GDALRasterAttributeTableH, int, int)
- double **GDALRATGetValueAsDouble** (GDALRasterAttributeTableH, int, int)
- void **GDALRATSetValueAsString** (GDALRasterAttributeTableH, int, int, const char *)
- void **GDALRATSetValueAsInt** (GDALRasterAttributeTableH, int, int, int)
- void **GDALRATSetValueAsDouble** (GDALRasterAttributeTableH, int, int, double)
- void **GDALRATSetRowCount** (GDALRasterAttributeTableH, int)
- CPLErr **GDALRATCreateColumn** (GDALRasterAttributeTableH, const char *, **GDALRATFieldType**, **GDALRATFieldUsage**)
- CPLErr **GDALRATSetLinearBinning** (GDALRasterAttributeTableH, double, double)
- int **GDALRATGetLinearBinning** (GDALRasterAttributeTableH, double *, double *)
- CPLErr **GDALRATInitializeFromColorTable** (GDALRasterAttributeTableH, GDALColorTableH)
- GDALColorTableH **GDALRATTranslateToColorTable** (int nEntryCount)
- void **GDALRATDumpReadable** (GDALRasterAttributeTableH, FILE *)
- GDALRasterAttributeTableH **GDALRATClone** (GDALRasterAttributeTableH)
- int **GDALRATGetRowOfValue** (GDALRasterAttributeTableH, double)
- void **GDALSetCacheMax** (int nBytes)
- int **GDALGetCacheMax** (void)
- int **GDALGetCacheUsed** (void)
- int **GDALFlushCacheBlock** (void)

44.11.1 Detailed Description

Public (C callable) GDAL entry points.

44.11.2 Define Documentation

44.11.2.1 #define GDAL_CHECK_VERSION(pszCallingComponentName) GDALCheckVersion(GDAL_VERSION_MAJOR, GDAL_VERSION_MINOR, pszCallingComponentName)

Helper macro for GDALCheckVersion

44.11.2.2 #define SRCVAL(papoSource, eSrcType, ii)

Value:

```
(eSrcType == GDT_Byte ? \
          ((GByte *)papoSource)[ii] : \
      (eSrcType == GDT_Float32 ? \
          ((float *)papoSource)[ii] : \
      (eSrcType == GDT_Float64 ? \
          ((double *)papoSource)[ii] : \
      (eSrcType == GDT_Int32 ? \
          ((GInt32 *)papoSource)[ii] : \
      (eSrcType == GDT_UInt16 ? \
          ((GUInt16 *)papoSource)[ii] : \
      (eSrcType == GDT_Int16 ? \
          ((GInt16 *)papoSource)[ii] : \
      (eSrcType == GDT_UInt32 ? \
          ((GUInt32 *)papoSource)[ii] : \
      (eSrcType == GDT_CInt16 ? \
          ((GInt16 *)papoSource)[ii * 2] : \
      (eSrcType == GDT_CInt32 ? \
          ((GInt32 *)papoSource)[ii * 2] : \
      (eSrcType == GDT_CFloat32 ? \
          ((float *)papoSource)[ii * 2] : \
      (eSrcType == GDT_CFloat64 ? \
          ((double *)papoSource)[ii * 2] : 0))))))))))))
```

SRCVAL - Macro which may be used by pixel functions to obtain a pixel from a source buffer.

44.11.3 Enumeration Type Documentation

44.11.3.1 enum GDALAccess

Flag indicating read/write, or read-only access to data.

Enumerator:

GA_ReadOnly Read only (no update) access

GA_Update Read/write access.

44.11.3.2 enum GDALColorInterp

Types of color interpretation for raster bands.

Enumerator:

GCI_GrayIndex Greyscale

GCI_PaletteIndex Paletted (see associated color table)
GCI_RedBand Red band of RGBA image
GCI_GreenBand Green band of RGBA image
GCI_BlueBand Blue band of RGBA image
GCI_AlphaBand Alpha (0=transparent, 255=opaque)
GCI_HueBand Hue band of HLS image
GCI_SaturationBand Saturation band of HLS image
GCI_LightnessBand Lightness band of HLS image
GCI_CyanBand Cyan band of CMYK image
GCI_MagentaBand Magenta band of CMYK image
GCI_YellowBand Yellow band of CMYK image
GCI_BlackBand Black band of CMLY image
GCI_YCbCr_YBand Y Luminance
GCI_YCbCr_CbBand Cb Chroma
GCI_YCbCr_CrBand Cr Chroma
GCI_Max Max current value

44.11.3.3 enum GDALDataType

Pixel data types

Enumerator:

GDT_Byte Eight bit unsigned integer
GDT_UInt16 Sixteen bit unsigned integer
GDT_Int16 Sixteen bit signed integer
GDT_UInt32 Thirty two bit unsigned integer
GDT_Int32 Thirty two bit signed integer
GDT_Float32 Thirty two bit floating point
GDT_Float64 Sixty four bit floating point
GDT_CInt16 Complex Int16
GDT_CInt32 Complex Int32
GDT_CFloat32 Complex Float32
GDT_CFloat64 Complex Float64

44.11.3.4 enum GDALPaletteInterp

Types of color interpretations for a **GDALColorTable** (p. **??**).

Enumerator:

GPI_Gray Grayscale (in **GDALColorEntry.c1** (p. **??**))
GPI_RGB Red, Green, Blue and Alpha in (in c1, c2, c3 and c4)
GPI_CMYK Cyan, Magenta, Yellow and Black (in c1, c2, c3 and c4)
GPI_HLS Hue, Lightness and Saturation (in c1, c2, and c3)

44.11.3.5 enum GDALRATFieldType

Enumerator:

GFT_Integer Integer field

GFT_Real Floating point (double) field

GFT_String String field

44.11.3.6 enum GDALRATFieldUsage

Enumerator:

GFU_Generic General purpose field.

GFU_PixelCount Histogram pixel count

GFU_Name Class name

GFU_Min Class range minimum

GFU_Max Class range maximum

GFU_MinMax Class value (min=max)

GFU_Red Red class color (0-255)

GFU_Green Green class color (0-255)

GFU_Blue Blue class color (0-255)

GFU_Alpha Alpha (0=transparent,255=opaque)

GFU_RedMin Color Range Red Minimum

GFU_GreenMin Color Range Green Minimum

GFU_BlueMin Color Range Blue Minimum

GFU_AlphaMin Color Range Alpha Minimum

GFU_RedMax Color Range Red Maximum

GFU_GreenMax Color Range Green Maximum

GFU_BlueMax Color Range Blue Maximum

GFU_AlphaMax Color Range Alpha Maximum

GFU_MaxCount Maximum GFU value

44.11.3.7 enum GDALRWFlag

Read/Write flag for RasterIO() method

Enumerator:

GF_Read Read data

GF_Write Write data

44.11.4 Function Documentation

44.11.4.1 CPLErr GDALAddBand (GDALDatasetH *hDataset*, GDALDataType *eType*, char ** *papszOptions*)

See also:

GDALDataset::AddBand() (p. **??**).

References GDALAddBand().

Referenced by GDALAddBand().

44.11.4.2 CPLErr GDALAddDerivedBandPixelFunc (const char * *pszFuncName*, GDALDerivedPixelFunc *pfnNewFunction*)

This adds a pixel function to the global list of available pixel functions for derived bands. Pixel functions must be registered in this way before a derived band tries to access data.

Derived bands are stored with only the name of the pixel function that it will apply, and if a pixel function matching the name is not found the IRasterIO() call will do nothing.

Parameters:

pszFuncName Name used to access pixel function

pfnNewFunction Pixel function associated with name. An existing pixel function registered with the same name will be replaced with the new one.

Returns:

CE_None, invalid (NULL) parameters are currently ignored.

44.11.4.3 void GDALAllRegister (void)

Register all known configured GDAL drivers.

This function will drive any of the following that are configured into GDAL. Possible others as well that haven't been updated in this documentation:

- GeoTIFF (GTiff)
- Geosoft GXF (GXF)
- Erdas Imagine (HFA)
- CEOS (CEOS)
- ELAS (ELAS)
- Arc/Info Binary Grid (AIGrid)
- SDTS Raster DEM (SDTS)
- OGDI (OGDI)
- ESRI Labelled BIL (EHdr)

- PCI .aux Labelled Raw Raster (PAux)
- HDF4 Hierachal Data Format Release 4
- HDF5 Hierachal Data Format Release 5
- GSAG Golden Software ASCII Grid
- GSBG Golden Software Binary Grid

References GDALDriverManager::AutoLoadDrivers(), GDALDriverManager::AutoSkipDrivers(), and GDALAllRegister().

Referenced by GDALAllRegister().

44.11.4.4 void GDALApplyGeoTransform (double * *padfGeoTransform*, double *dfPixel*, double *dfLine*, double * *pdfGeoX*, double * *pdfGeoY*)

Apply GeoTransform to x/y coordinate.

Applies the following computation, converting a (pixel,line) coordinate into a georeferenced (geo_x,geo_y) location.

*pdfGeoX = padfGeoTransform[0] + dfPixel * padfGeoTransform[1] + dfLine * padfGeoTransform[2];
*pdfGeoY = padfGeoTransform[3] + dfPixel * padfGeoTransform[4] + dfLine * padfGeoTransform[5];

Parameters:

padfGeoTransform Six coefficient GeoTransform to apply.

dfPixel Input pixel position.

dfLine Input line position.

pdfGeoX output location where GeoX (easting/longitude) location is placed.

pdfGeoY output location where GeoX (northing/latitude) location is placed.

References GDALApplyGeoTransform().

Referenced by GDALApplyGeoTransform().

44.11.4.5 CPLErr GDALBuildOverviews (GDALDatasetH *hDataset*, const char * *pszResampling*, int *nOverviews*, int * *panOverviewList*, int *nListBands*, int * *panBandList*, GDALProgressFunc *pfnProgress*, void * *pProgressData*)

See also:

GDALDataset::BuildOverviews() (p. **??**)

References GDALBuildOverviews().

Referenced by GDALBuildOverviews().

44.11.4.6 int GDALCheckVersion (int *nVersionMajor*, int *nVersionMinor*, const char * *pszCallingComponentName*)

Return TRUE if GDAL library version at runtime matches nVersionMajor.nVersionMinor.

The purpose of this method is to ensure that calling code will run with the GDAL version it is compiled for. It is primarly intented for external plugins.

Parameters:

nVersionMajor Major version to be tested against

nVersionMinor Minor version to be tested against

pszCallingComponentName If not NULL, in case of version mismatch, the method will issue a failure mentionning the name of the calling component.

References GDALCheckVersion().

Referenced by GDALCheckVersion().

44.11.4.7 void GDALClose (GDALDatasetH *hDS*)

Close GDAL dataset.

For non-shared datasets (opened with **GDALOpen()** (p. **??**)) the dataset is closed using the C++ "delete" operator, recovering all dataset related resources. For shared datasets (opened with **GDALOpenShared()** (p. **??**)) the dataset is dereferenced, and closed only if the referenced count has dropped below 1.

Parameters:

hDS The dataset to close. May be cast from a "GDALDataset *".

References GDALDataset::Dereference(), GDALClose(), and GDALDataset::GetShared().

Referenced by GDALDriver::CopyFiles(), GDALDriver::Delete(), GDALClose(), and GDALDriver::Rename().

44.11.4.8 void GDALComputeRasterMinMax (GDALRasterBandH *hBand*, int *bApproxOK*, double *adfMinMax*[2])

Compute the min/max values for a band.

If approximate is OK, then the band's GetMinimum()/GetMaximum() will be trusted. If it doesn't work, a subsample of blocks will be read to get an approximate min/max. If the band has a nodata value it will be excluded from the minimum and maximum.

If bApprox is FALSE, then all pixels will be read and used to compute an exact range.

This method is the same as the method GDALRasterBand::ComputeRasterMinMax().

Parameters:

hBand the band to operate on.

bApproxOK TRUE if an approximate (faster) answer is OK, otherwise FALSE.

adfMinMax the array in which the minimum (adfMinMax[0]) and the maximum (adfMinMax[1]) are returned.

Returns:

CE_None on success or CE_Failure on failure.

References GDALComputeRasterMinMax().

Referenced by GDALComputeRasterMinMax().

44.11.4.9 void GDALCopyBits (const GByte * *pabySrcData*, int *nSrcOffset*, int *nSrcStep*, GByte * *pabyDstData*, int *nDstOffset*, int *nDstStep*, int *nBitCount*, int *nStepCount*)

Bitwise word copying.

A function for moving sets of partial bytes around. Loosely speaking this is a bitswise analog to **GDALCopyWords()** (p. **??**).

It copies nStepCount "words" where each word is nBitCount bits long. The nSrcStep and nDstStep are the number of bits from the start of one word to the next (same as nBitCount if they are packed). The nSrcOffset and nDstOffset are the offset into the source and destination buffers to start at, also measured in bits.

All bit offsets are assumed to start from the high order bit in a byte (ie. most significant bit first). Currently this function is not very optimized, but it may be improved for some common cases in the future as needed.

Parameters:

pabySrcData the source data buffer.

nSrcOffset the offset (in bits) in pabySrcData to the start of the first word to copy.

nSrcStep the offset in bits from the start one source word to the start of the next.

pabyDstData the destination data buffer.

nDstOffset the offset (in bits) in pabyDstData to the start of the first word to copy over.

nDstStep the offset in bits from the start one word to the start of the next.

nBitCount the number of bits in a word to be copied.

nStepCount the number of words to copy.

References GDALCopyBits().

Referenced by GDALCopyBits().

44.11.4.10 CPLErr GDALCopyDatasetFiles (GDALDriverH *hDriver*, const char * *pszNewName*, const char * *pszOldName*)

See also:

GDALDriver::CopyFiles() (p. **??**)

References GDALCopyDatasetFiles(), and GDALIdentifyDriver().

Referenced by GDALCopyDatasetFiles().

44.11.4.11 void GDALCopyWords (void * *pSrcData*, GDALDataType *eSrcType*, int *nSrcPixelOffset*, void * *pDstData*, GDALDataType *eDstType*, int *nDstPixelOffset*, int *nWordCount*)

Copy pixel words from buffer to buffer.

This function is used to copy pixel word values from one memory buffer to another, with support for conversion between data types, and differing step factors. The data type conversion is done using the normal GDAL rules. Values assigned to a lower range integer type are clipped. For instance assigning GDT_Int16 values to a GDT_Byte buffer will cause values less the 0 to be set to 0, and values larger than 255 to be set to 255. Assignment from floating point to integer uses default C type casting semantics. Assignment from non-complex to complex will result in the imaginary part being set to zero on output. Assigment

from complex to non-complex will result in the complex portion being lost and the real component being preserved (*not magnitidue!*).

No assumptions are made about the source or destination words occuring on word boundaries. It is assumed that all values are in native machine byte order.

Parameters:

pSrcData

References GDALCopyWords(), GDALGetDataTypeSize(), GDT_Byte, GDT_CFloat32, GDT_CFloat64, GDT_CInt16, GDT_CInt32, GDT_Float32, GDT_Float64, GDT_Int16, GDT_Int32, GDT_UInt16, and GDT_UInt32.

Referenced by GDALRasterBand::Fill(), GDALCopyWords(), and GDALWarpOperation::WarpRegion().

44.11.4.12 GDALDatasetH GDALCreate (GDALDriverH *hDriver*, const char * *pszFilename*, int *nXSize*, int *nYSize*, int *nBands*, GDALDataType *eBandType*, char ** *papszOptions*)

See also:

GDALDriver::Create() (p. **??**)

References GDALDriver::Create(), and GDALCreate().

Referenced by GDALCreate().

44.11.4.13 GDALDatasetH GDALCreateCopy (GDALDriverH *hDriver*, const char * *pszFilename*, GDALDatasetH *hSrcDS*, int *bStrict*, char ** *papszOptions*, GDALProgressFunc *pfnProgress*, void * *pProgressData*)

See also:

GDALDriver::CreateCopy() (p. **??**)

References GDALDriver::CreateCopy(), and GDALCreateCopy().

Referenced by GDALCreateCopy().

44.11.4.14 void* GDALCreateScaledProgress (double *dfMin*, double *dfMax*, GDALProgressFunc *pfnProgress*, void * *pData*)

Create scaled progress transformer.

Sometimes when an operations wants to report progress it actually invokes several subprocesses which also take GDALProgressFunc()s, and it is desirable to map the progress of each sub operation into a portion of 0.0 to 1.0 progress of the overall process. The scaled progress function can be used for this.

For each subsection a scaled progress function is created and instead of passing the overall progress func down to the sub functions, the **GDALScaledProgress()** (p. **??**) function is passed instead.

Parameters:

dfMin the value to which 0.0 in the sub operation is mapped.

dfMax the value to which 1.0 is the sub operation is mapped.

pfnProgress the overall progress function.

pData the overall progress function callback data.

Returns:

pointer to pass as pProgressArg to sub functions. Should be freed with **GDALDestroyScaledProgress()** (p. **??**).

Example:

```
int MyOperation( ..., GDALProgressFunc pfnProgress, void *pProgressData );

{
    void *pScaledProgress;

    pScaledProgress = GDALCreateScaledProgress( 0.0, 0.5, pfnProgress,
                                                pProgressData );
    GDALDoLongSlowOperation( ..., GDALScaledProgress, pScaledProgress );
    GDALDestroyScaledProgress( pScaledProgress );

    pScaledProgress = GDALCreateScaledProgress( 0.5, 1.0, pfnProgress,
                                                pProgressData );
    GDALDoAnotherOperation( ..., GDALScaledProgress, pScaledProgress );
    GDALDestroyScaledProgress( pScaledProgress );

    return ...;
}
```

References GDALCreateScaledProgress().

Referenced by GDALCreateScaledProgress().

44.11.4.15 CPLErr GDALDatasetCopyWholeRaster (GDALDatasetH *hSrcDS*, GDALDatasetH *hDstDS*, char ** *papszOptions*, GDALProgressFunc *pfnProgress*, void * *pProgressData*)

Copy all dataset raster data.

This function copies the complete raster contents of one dataset to another similarly configured dataset. The source and destination dataset must have the same number of bands, and the same width and height. The bands do not have to have the same data type.

This function is primarily intended to support implementation of driver specific CreateCopy() functions. It implements efficient copying, in particular "chunking" the copy in substantial blocks and, if appropriate, performing the transfer in a pixel interleaved fashion.

Currently the only papszOptions value supported is "INTERLEAVE=PIXEL" to force pixel interleaved operation. More options may be supported in the future.

Parameters:

hSrcDS the source dataset

hDstDS the destination dataset

papszOptions transfer hints in "StringList" Name=Value format.

pfnProgress progress reporting function.

pProgressData callback data for progress function.

Returns:

CE_None on success, or CE_Failure on failure.

References GDALDatasetCopyWholeRaster(), GDALGetCacheMax(), GDALGetDataTypeSize(), GDALRasterBand::GetBlockSize(), GDALMajorObject::GetMetadataItem(), GDALDataset::GetRasterBand(), GDALDataset::GetRasterCount(), GDALRasterBand::GetRasterDataType(), GDALDataset::GetRasterXSize(), GDALDataset::GetRasterYSize(), GF_Read, GF_Write, GDALDataset::RasterIO(), and VSIMalloc3().

Referenced by GDALDatasetCopyWholeRaster().

44.11.4.16 CPLErr GDALDatasetRasterIO (GDALDatasetH *hDS*, GDALRWFlag *eRWFlag*, int *nXOff*, int *nYOff*, int *nXSize*, int *nYSize*, void * *pData*, int *nBufXSize*, int *nBufYSize*, GDALDataType *eBufType*, int *nBandCount*, int * *panBandMap*, int *nPixelSpace*, int *nLineSpace*, int *nBandSpace*)

See also:

GDALDataset::RasterIO() (p. **??**)

References GDALDatasetRasterIO(), and GDALDataset::RasterIO().

Referenced by GDALDatasetRasterIO(), GDALWarpOperation::WarpRegion(), and GDALWarpOperation::WarpRegionToBuffer().

44.11.4.17 int GDALDataTypeIsComplex (GDALDataType *eDataType*)

Is data type complex?

Returns:

TRUE if the passed type is complex (one of GDT_CInt16, GDT_CInt32, GDT_CFloat32 or GDT_CFloat64), that is it consists of a real and imaginary component.

References GDALDataTypeIsComplex(), GDT_CFloat32, GDT_CFloat64, GDT_CInt16, and GDT_CInt32.

Referenced by GDALChecksumImage(), GDALDataTypeIsComplex(), GDALDataTypeUnion(), GDALRegenerateOverviews(), and GDALWarpOperation::Initialize().

44.11.4.18 GDALDataType GDALDataTypeUnion (GDALDataType *eType1*, GDALDataType *eType2*)

Return the smallest data type that can fully express both input data types.

Parameters:

eType1

eType2

Returns:

a data type able to express eType1 and eType2.

References GDALDataTypeIsComplex(), GDALDataTypeUnion(), GDT_Byte, GDT_CFloat32, GDT_CFloat64, GDT_CInt16, GDT_CInt32, GDT_Float32, GDT_Float64, GDT_Int16, GDT_Int32, GDT_UInt16, and GDT_UInt32.

Referenced by GDALDataTypeUnion(), and GDALWarpOperation::Initialize().

44.11.4.19 double GDALDecToPackedDMS (double *dfDec*)

Convert decimal degrees into packed DMS value (DDDMMMSSS.SS).

See **CPLDecToPackedDMS()** (p. **??**).

References GDALDecToPackedDMS().

Referenced by GDALDecToPackedDMS().

44.11.4.20 CPLErr GDALDeleteDataset (GDALDriverH *hDriver*, const char * *pszFilename*)

See also:

GDALDriver::Delete() (p. **??**)

References GDALDeleteDataset(), and GDALIdentifyDriver().

Referenced by GDALDeleteDataset().

44.11.4.21 int GDALDereferenceDataset (GDALDatasetH *hDataset*)

See also:

GDALDataset::Dereference() (p. **??**)

References GDALDereferenceDataset().

Referenced by GDALDereferenceDataset().

44.11.4.22 void GDALDeregisterDriver (GDALDriverH *hDriver*)

See also:

GDALDriverManager::GetDeregisterDriver()

References GDALDriverManager::DeregisterDriver(), and GDALDeregisterDriver().

Referenced by GDALDeregisterDriver().

44.11.4.23 void GDALDestroyDriver (GDALDriverH *hDriver*)

Destroy a **GDALDriver** (p. **??**).

This is roughly equivelent to deleting the driver, but is guaranteed to take place in the GDAL heap. It is important this that function not be called on a driver that is registered with the **GDALDriverManager** (p. **??**).

Parameters:

hDriver the driver to destroy.

References GDALDestroyDriver().

Referenced by GDALDestroyDriver().

44.11.4.24 void GDALDestroyDriverManager (void)

Destroy the driver manager.

Incidently unloads all managed drivers.

NOTE: This function is not thread safe. It should not be called while other threads are actively using GDAL.

References GDALDestroyDriverManager().

Referenced by GDALDestroyDriverManager().

44.11.4.25 void GDALDestroyScaledProgress (void * *pData*)

Cleanup scaled progress handle.

This function cleans up the data associated with a scaled progress function as returned by GADLCreateScaledProgress().

Parameters:

pData scaled progress handle returned by **GDALCreateScaledProgress()** (p. **??**).

References GDALDestroyScaledProgress().

Referenced by GDALDestroyScaledProgress().

44.11.4.26 int GDALDumpOpenDatasets (FILE * *fp*)

List open datasets.

Dumps a list of all open datasets (shared or not) to the indicated text file (may be stdout or stderr). This function is primariliy intended to assist in debugging "dataset leaks" and reference counting issues. The information reported includes the dataset name, referenced count, shared status, driver name, size, and band count.

References GDALDumpOpenDatasets().

Referenced by GDALDumpOpenDatasets().

44.11.4.27 CPLErr GDALFillRaster (GDALRasterBandH *hBand*, double *dfRealValue*, double *dfImaginaryValue*)

Fill this band with a constant value. Set *dfImaginaryValue* to zero non-complex rasters.

Parameters:

dfRealvalue Real component of fill value

dfImaginaryValue Imaginary component of fill value

See also:

GDALRasterBand::Fill() (p. **??**)

Returns:

CE_Failure if the write fails, otherwise CE_None

References GDALFillRaster().

Referenced by GDALFillRaster().

44.11.4.28 void GDALFlushCache (GDALDatasetH *hDS*)

See also:

GDALDataset::FlushCache() (p. **??**).

References GDALFlushCache().

Referenced by GDALFlushCache(), and GDALWarpOperation::WarpRegion().

44.11.4.29 CPLErr GDALFlushRasterCache (GDALRasterBandH *hBand*)

See also:

GDALRasterBand::FlushCache() (p. **??**)

References GDALFlushRasterCache().

Referenced by GDALFlushRasterCache().

44.11.4.30 int GDALGCPsToGeoTransform (int *nGCPCount*, const GDAL_GCP * *pasGCPs*, double * *padfGeoTransform*, int *bApproxOK*)

Generate Geotransform from GCPs.

Given a set of GCPs perform first order fit as a geotransform.

Due to imprecision in the calculations the fit algorithm will often return non-zero rotational coefficients even if given perfectly non-rotated inputs. A special case has been implemented for corner corner coordinates given in TL, TR, BR, BL order. So when using this to get a geotransform from 4 corner coordinates, pass them in this order.

Parameters:

nGCPCount the number of GCPs being passed in.

pasGCPs the list of GCP structures.

padfGeoTransform the six double array in which the affine geotransformation will be returned.

bApproxOK If FALSE the function will fail if the geotransform is not essentially an exact fit (within 0.25 pixel) for all GCPs.

Returns:

TRUE on success or FALSE if there aren't enough points to prepare a geotransform, the pointers are ill-determined or if bApproxOK is FALSE and the fit is poor.

References GDAL_GCP::dfGCPLine, GDAL_GCP::dfGCPPixel, GDAL_GCP::dfGCPX, GDAL_GCP::dfGCPY, and GDALGCPsToGeoTransform().

Referenced by GDALGCPsToGeoTransform().

44.11.4.31 int GDALGeneralCmdLineProcessor (int *nArgc*, char ∗∗∗ *ppapszArgv*, int *nOptions*)

General utility option processing.

This function is intended to provide a variety of generic commandline options for all GDAL commandline utilities. It takes care of the following commandline options:

–version: report version of GDAL in use. –license: report GDAL license info. –formats: report all format drivers configured. –format [format]: report details of one format driver. –optfile filename: expand an option file into the argument list. –config key value: set system configuration option. –debug [on/off/value]: set debug level. –mempreload dir: preload directory contents into /vsimem –help-general: report detailed help on general options.

The argument array is replaced "in place" and should be freed with **CSLDestroy()** (p. **??**) when no longer needed. The typical usage looks something like the following. Note that the formats should be registered so that the –formats and –format options will work properly.

int main(int argc, char ∗∗ argv) { **GDALAllRegister()** (p. **??**);

argc = GDALGeneralCmdLineProcessor(argc, &argv, 0); if(argc < 1) exit(-argc);

Parameters:

nArgc number of values in the argument list.

Pointer to the argument list array (will be updated in place).

Returns:

updated nArgc argument count. Return of 0 requests terminate without error, return of -1 requests exit with error code.

References CPLFormFilename(), GDALGeneralCmdLineProcessor(), GDALGetDriver(), GDALGetDriverByName(), GDALGetDriverCount(), GDALGetDriverLongName(), GDALGetDriverShortName(), GDALGetMetadata(), GDALGetMetadataItem(), and GDALVersionInfo().

Referenced by GDALGeneralCmdLineProcessor().

44.11.4.32 int GDALGetAccess (GDALDatasetH *hDS*)

See also:

GDALDataset::GetAccess()

References GDALGetAccess().

Referenced by GDALGetAccess().

44.11.4.33 GDALDatasetH GDALGetBandDataset (GDALRasterBandH *hBand*)

See also:

GDALRasterBand::GetDataset() (p. **??**)

References GDALGetBandDataset().

Referenced by GDALContourGenerate(), GDALGetBandDataset(), and GDALPolygonize().

44.11.4.34 int GDALGetBandNumber (GDALRasterBandH *hBand*)

See also:

GDALRasterBand::GetBand() (p. **??**)

References GDALGetBandNumber().

Referenced by GDALGetBandNumber().

44.11.4.35 void GDALGetBlockSize (GDALRasterBandH *hBand*, int * *pnXSize*, int * *pnYSize*)

See also:

GDALRasterBand::GetBlockSize() (p. **??**)

References GDALGetBlockSize(), and GDALRasterBand::GetBlockSize().

Referenced by GDALGetBlockSize().

44.11.4.36 int GDALGetCacheMax (void)

Get maximum cache memory.

Gets the maximum amount of memory available to the GDALRasterBlock caching system for caching GDAL read/write imagery.

Returns:

maximum in bytes.

References GDALGetCacheMax().

Referenced by GDALDatasetCopyWholeRaster(), and GDALGetCacheMax().

44.11.4.37 int GDALGetCacheUsed (void)

Get cache memory used.

Returns:

the number of bytes of memory currently in use by the GDALRasterBlock memory caching.

References GDALGetCacheUsed().

Referenced by GDALGetCacheUsed().

44.11.4.38 const char* GDALGetColorInterpretationName (GDALColorInterp)

Translate a GDALColorInterp into a user displayable string.

References GCI_AlphaBand, GCI_BlackBand, GCI_BlueBand, GCI_CyanBand, GCI_GrayIndex, GCI_GreenBand, GCI_HueBand, GCI_LightnessBand, GCI_MagentaBand, GCI_PaletteIndex, GCI_-RedBand, GCI_SaturationBand, GCI_YCbCr_CbBand, GCI_YCbCr_CrBand, GCI_YCbCr_YBand, GCI_YellowBand, and GDALGetColorInterpretationName().

Referenced by GDALGetColorInterpretationName().

44.11.4.39 GDALDriverH GDALGetDatasetDriver (GDALDatasetH *hDataset*)

See also:

GDALDataset::GetDriver() (p. **??**)

References GDALGetDatasetDriver().

Referenced by GDALGetDatasetDriver().

44.11.4.40 GDALDataType GDALGetDataTypeByName (const char * *pszName*)

Get data type by symbolic name.

Returns a data type corresponding to the given symbolic name. This function is opposite to the **GDALGetDataTypeName()** (p. **??**).

Parameters:

pszName string containing the symbolic name of the type.

Returns:

GDAL data type.

References GDALGetDataTypeByName(), and GDALGetDataTypeName().

Referenced by GDALGetDataTypeByName().

44.11.4.41 const char* GDALGetDataTypeName (GDALDataType *eDataType*)

Get name of data type.

Returns a symbolic name for the data type. This is essentially the the enumerated item name with the GDT_ prefix removed. So GDT_Byte returns "Byte". The returned strings are static strings and should not be modified or freed by the application. These strings are useful for reporting datatypes in debug statements, errors and other user output.

Parameters:

eDataType type to get name of.

Returns:

string corresponding to type.

References GDALGetDataTypeName(), GDT_Byte, GDT_CFloat32, GDT_CFloat64, GDT_CInt16, GDT_CInt32, GDT_Float32, GDT_Float64, GDT_Int16, GDT_Int32, GDT_UInt16, and GDT_UInt32.

Referenced by GDALDriver::Create(), GDALGetDataTypeByName(), and GDALGetDataTypeName().

44.11.4.42 int GDALGetDataTypeSize (GDALDataType *eDataType*)

Get data type size in bits.

Returns the size of a a GDT_* type in bits, **not bytes**!

Parameters:

data type, such as GDT_Byte.

Returns:

the number of bits or zero if it is not recognised.

References GDALGetDataTypeSize(), GDT_Byte, GDT_CFloat32, GDT_CFloat64, GDT_CInt16, GDT_CInt32, GDT_Float32, GDT_Float64, GDT_Int16, GDT_Int32, GDT_UInt16, and GDT_UInt32.

Referenced by GDALRasterBand::ComputeStatistics(), GDALRasterBand::Fill(), GDALCopyWords(), GDALDatasetCopyWholeRaster(), GDALGetDataTypeSize(), GDALRasterizeGeometries(), GDALRegenerateOverviews(), GDALRasterBand::GetHistogram(), GDALRasterBand::RasterIO(), GDALDataset::RasterIO(), GDALWarpOperation::WarpRegion(), and GDALWarpOperation::WarpRegionToBuffer().

44.11.4.43 const char* GDALGetDescription (GDALMajorObjectH *hObject*)

See also:

GDALMajorObject::GetDescription() (p. **??**)

References GDALGetDescription().

Referenced by GDALCreateGenImgProjTransformer2(), and GDALGetDescription().

44.11.4.44 GDALDriverH GDALGetDriver (int *iDriver*)

See also:

GDALDriverManager::GetDriver() (p. **??**)

References GDALGetDriver(), and GDALDriverManager::GetDriver().

Referenced by GDALGeneralCmdLineProcessor(), and GDALGetDriver().

44.11.4.45 GDALDriverH GDALGetDriverByName (const char * *pszName*)

See also:

GDALDriverManager::GetDriverByName() (p. **??**)

References GDALGetDriverByName(), and GDALDriverManager::GetDriverByName().

Referenced by GDALGeneralCmdLineProcessor(), and GDALGetDriverByName().

44.11.4.46 int GDALGetDriverCount (void)

See also:

GDALDriverManager::GetDriverCount() (p. **??**)

References GDALGetDriverCount(), and GDALDriverManager::GetDriverCount().

Referenced by GDALGeneralCmdLineProcessor(), and GDALGetDriverCount().

44.11.4.47 const char* GDALGetDriverCreationOptionList (GDALDriverH *hDriver*)

Return the list of creation options of the driver

Return the list of creation options of the driver used by Create() and CreateCopy() as an XML string

Parameters:

hDriver the handle of the driver

Returns:

an XML string that describes the list of creation options or empty string. The returned string should not be freed and is owned by the driver.

References GDALGetDriverCreationOptionList().

Referenced by GDALGetDriverCreationOptionList().

44.11.4.48 const char* GDALGetDriverHelpTopic (GDALDriverH *hDriver*)

Return the URL to the help that describes the driver

Return the URL to the help that describes the driver. That URL is relative to the GDAL documentation directory.

For the GeoTIFF driver, this is "frmt_gtiff.html"

Parameters:

hDriver the handle of the driver

Returns:

the URL to the help that describes the driver or NULL. The returned string should not be freed and is owned by the driver.

References GDALGetDriverHelpTopic().

Referenced by GDALGetDriverHelpTopic().

44.11.4.49 const char* GDALGetDriverLongName (GDALDriverH *hDriver*)

Return the long name of a driver

Return the long name of a the driver.

For the GeoTIFF driver, this is "GeoTIFF"

Parameters:

hDriver the handle of the driver

Returns:

the long name of the driver or empty string. The returned string should not be freed and is owned by the driver.

References GDALGetDriverLongName().

Referenced by GDALGeneralCmdLineProcessor(), and GDALGetDriverLongName().

44.11.4.50 const char* GDALGetDriverShortName (GDALDriverH *hDriver*)

Return the short name of a driver

Return the short name of a the driver. This is the string that can be passed to the **GDALGetDriverByName()** (p. **??**) function.

For the GeoTIFF driver, this is "GTiff"

Parameters:

hDriver the handle of the driver

Returns:

the short name of the driver. The returned string should not be freed and is owned by the driver.

References GDALGetDriverShortName().

Referenced by GDALGeneralCmdLineProcessor(), and GDALGetDriverShortName().

44.11.4.51 char** GDALGetFileList (GDALDatasetH *hDS*)

See also:

GDALDataset::GetFileList() (p. **??**)

References GDALGetFileList().

Referenced by GDALDriver::CopyFiles(), GDALDriver::Delete(), GDALGetFileList(), and GDALDriver::Rename().

44.11.4.52 int GDALGetGCPCount (GDALDatasetH *hDS*)

See also:

GDALDataset::GetGCPCount() (p. **??**)

References GDALGetGCPCount().

Referenced by GDALAutoCreateWarpedVRT(), GDALCreateGenImgProjTransformer2(), and GDALGetGCPCount().

44.11.4.53 const char* GDALGetGCPProjection (GDALDatasetH *hDS*)

See also:

GDALDataset::GetGCPProjection() (p. **??**)

References GDALGetGCPProjection().

Referenced by GDALAutoCreateWarpedVRT(), GDALCreateGenImgProjTransformer2(), and GDALGetGCPProjection().

44.11.4.54 const GDAL_GCP* GDALGetGCPs (GDALDatasetH *hDS*)

See also:

GDALDataset::GetGCPs() (p. **??**)

References GDALGetGCPs().

Referenced by GDALCreateGenImgProjTransformer2(), and GDALGetGCPs().

44.11.4.55 CPLErr GDALGetGeoTransform (GDALDatasetH *hDS*, double * *padfTransform*)

See also:

GDALDataset::GetGeoTransform() (p. **??**)

References GDALGetGeoTransform(), and GDALDataset::GetGeoTransform().

Referenced by GDALContourGenerate(), GDALCreateGenImgProjTransformer2(), GDALGetGeoTransform(), and GDALPolygonize().

44.11.4.56 void* GDALGetInternalHandle (GDALDatasetH *hDS*, const char * *pszRequest*)

See also:

GDALDataset::GetInternalHandle() (p. **??**)

References GDALGetInternalHandle(), and GDALDataset::GetInternalHandle().

Referenced by GDALGetInternalHandle().

44.11.4.57 char** GDALGetMetadata (GDALMajorObjectH *hObject*, const char * *pszDomain*)

See also:

GDALMajorObject::GetMetadata() (p. **??**)

References GDALGetMetadata().

Referenced by GDALCreateGenImgProjTransformer2(), GDALGeneralCmdLineProcessor(), and GDALGetMetadata().

44.11.4.58 const char* GDALGetMetadataItem (GDALMajorObjectH *hObject*, const char * *pszName*, const char * *pszDomain*)

See also:

GDALMajorObject::GetMetadataItem() (p. **??**)

References GDALGetMetadataItem().

Referenced by GDALGeneralCmdLineProcessor(), and GDALGetMetadataItem().

44.11.4.59 void GDALGetOpenDatasets (GDALDatasetH ** *ppahDSList*, int * *pnCount*)

See also:

GDALDataset::GetOpenDatasets() (p. **??**)

References GDALGetOpenDatasets(), and GDALDataset::GetOpenDatasets().

Referenced by GDALGetOpenDatasets().

44.11.4.60 GDALRasterBandH GDALGetOverview (GDALRasterBandH *hBand*, int *i*)

See also:

GDALRasterBand::GetOverview() (p. **??**)

References GDALGetOverview().

Referenced by GDALGetOverview().

44.11.4.61 int GDALGetOverviewCount (GDALRasterBandH *hBand*)

See also:

GDALRasterBand::GetOverviewCount() (p. **??**)

References GDALGetOverviewCount().

Referenced by GDALGetOverviewCount().

44.11.4.62 const char* GDALGetPaletteInterpretationName (GDALPaletteInterp)

Translate a GDALPaletteInterp into a user displayable string.

References GDALGetPaletteInterpretationName(), GPI_CMYK, GPI_Gray, GPI_HLS, and GPI_RGB.

Referenced by GDALGetPaletteInterpretationName().

44.11.4.63 const char* GDALGetProjectionRef (GDALDatasetH *hDS*)

See also:

GDALDataset::GetProjectionRef() (p. **??**)

References GDALGetProjectionRef(), and GDALDataset::GetProjectionRef().

Referenced by GDALAutoCreateWarpedVRT(), GDALCreateGenImgProjTransformer2(), and GDALGetProjectionRef().

44.11.4.64 GDALAccess GDALGetRasterAccess (GDALRasterBandH *hBand*)

See also:

GDALRasterBand::GetAccess() (p. **??**)

References GA_ReadOnly, and GDALGetRasterAccess().

Referenced by GDALGetRasterAccess().

44.11.4.65 GDALRasterBandH GDALGetRasterBand (GDALDatasetH *hDS*, int *nBandId*)

See also:

GDALDataset::GetRasterBand() (p. **??**).

References GDALGetRasterBand(), and GDALDataset::GetRasterBand().

Referenced by GDALCreateWarpedVRT(), GDALGetRasterBand(), GDALReprojectImage(), GDALSimpleImageWarp(), and GDALWarpOperation::Initialize().

44.11.4.66 int GDALGetRasterBandXSize (GDALRasterBandH *hBand*)

See also:

GDALRasterBand::GetXSize() (p. **??**)

References GDALGetRasterBandXSize().

Referenced by GDALComputeMedianCutPCT(), GDALContourGenerate(), GDALDitherRGB2PCT(), GDALGetRasterBandXSize(), GDALPolygonize(), and GDALSieveFilter().

44.11.4.67 int GDALGetRasterBandYSize (GDALRasterBandH *hBand*)

See also:

GDALRasterBand::GetYSize() (p. **??**)

References GDALGetRasterBandYSize().

Referenced by GDALComputeMedianCutPCT(), GDALContourGenerate(), GDALDitherRGB2PCT(), GDALGetRasterBandYSize(), GDALPolygonize(), and GDALSieveFilter().

44.11.4.68 char** GDALGetRasterCategoryNames (GDALRasterBandH *hBand*)

See also:

GDALRasterBand::GetCategoryNames() (p. **??**)

References GDALGetRasterCategoryNames().

Referenced by GDALGetRasterCategoryNames().

44.11.4.69 GDALColorInterp GDALGetRasterColorInterpretation (GDALRasterBandH *hBand*)

See also:

GDALRasterBand::GetColorInterpretation() (p. **??**)

References GDALGetRasterColorInterpretation().

Referenced by GDALGetRasterColorInterpretation().

44.11.4.70 GDALColorTableH GDALGetRasterColorTable (GDALRasterBandH *hBand*)

See also:

GDALRasterBand::GetColorTable() (p. **??**)

References GDALGetRasterColorTable().

Referenced by GDALGetRasterColorTable().

44.11.4.71 int GDALGetRasterCount (GDALDatasetH *hDS*)

See also:

GDALDataset::GetRasterCount() (p. **??**).

References GDALGetRasterCount(), and GDALDataset::GetRasterCount().

Referenced by GDALAutoCreateWarpedVRT(), GDALGetRasterCount(), GDALReprojectImage(), GDALSimpleImageWarp(), and GDALWarpOperation::Initialize().

44.11.4.72 GDALDataType GDALGetRasterDataType (GDALRasterBandH *hBand*)

See also:

GDALRasterBand::GetRasterDataType() (p. **??**)

References GDALGetRasterDataType(), and GDALRasterBand::GetRasterDataType().

Referenced by GDALChecksumImage(), GDALGetRasterDataType(), and GDALWarpOperation::Initialize().

44.11.4.73 CPLErr GDALGetRasterHistogram (GDALRasterBandH *hBand*, double *dfMin*, double *dfMax*, int *nBuckets*, int * *panHistogram*, int *bIncludeOutOfRange*, int *bApproxOK*, GDALProgressFunc *pfnProgress*, void * *pProgressData*)

See also:

GDALRasterBand::GetHistogram() (p. **??**)

References GDALGetRasterHistogram(), and GDALRasterBand::GetHistogram().

Referenced by GDALGetRasterHistogram().

44.11.4.74 double GDALGetRasterMaximum (GDALRasterBandH *hBand*, int * *pbSuccess*)

See also:

GDALRasterBand::GetMaximum() (p. **??**)

References GDALGetRasterMaximum().

Referenced by GDALGetRasterMaximum().

44.11.4.75 double GDALGetRasterMinimum (GDALRasterBandH *hBand*, int * *pbSuccess*)

See also:

GDALRasterBand::GetMinimum() (p. **??**)

References GDALGetRasterMinimum().

Referenced by GDALGetRasterMinimum().

44.11.4.76 double GDALGetRasterNoDataValue (GDALRasterBandH *hBand*, int * *pbSuccess*)

See also:

GDALRasterBand::GetNoDataValue() (p. **??**)

References GDALGetRasterNoDataValue().

Referenced by GDALGetRasterNoDataValue(), and GDALReprojectImage().

44.11.4.77 const char* GDALGetRasterUnitType (GDALRasterBandH *hBand*)

See also:

GDALRasterBand::GetUnitType() (p. **??**)

References GDALGetRasterUnitType().

Referenced by GDALGetRasterUnitType().

44.11.4.78 int GDALGetRasterXSize (GDALDatasetH *hDataset*)

See also:

GDALDataset::GetRasterXSize() (p. **??**).

References GDALGetRasterXSize().

Referenced by GDALGetRasterXSize(), GDALReprojectImage(), GDALSimpleImageWarp(), and GDALSuggestedWarpOutput2().

44.11.4.79 int GDALGetRasterYSize (GDALDatasetH *hDataset*)

See also:

GDALDataset::GetRasterYSize() (p. **??**).

References GDALGetRasterYSize().

Referenced by GDALGetRasterYSize(), GDALReprojectImage(), GDALSimpleImageWarp(), and GDALSuggestedWarpOutput2().

44.11.4.80 int GDALHasArbitraryOverviews (GDALRasterBandH *hBand*)

See also:

GDALRasterBand::HasArbitraryOverviews() (p. **??**)

References GDALHasArbitraryOverviews().

Referenced by GDALHasArbitraryOverviews().

44.11.4.81 GDALDriverH GDALIdentifyDriver (const char * *pszFilename*, char ** *papszFileList*)

Identify the driver that can open a raster file.

This function will try to identify the driver that can open the passed file name by invoking the Identify method of each registered **GDALDriver** (p. **??**) in turn. The first driver that successful identifies the file name will be returned. If all drivers fail then NULL is returned.

In order to reduce the need for such searches touch the operating system file system machinery, it is possible to give an optional list of files. This is the list of all files at the same level in the file system as the target file, including the target file. The filenames will not include any path components, are an essentially just the output of CPLReadDir() on the parent directory. If the target object does not have filesystem semantics then the file list should be NULL.

Parameters:

pszFilename the name of the file to access. In the case of exotic drivers this may not refer to a physical file, but instead contain information for the driver on how to access a dataset.

papszFileList an array of strings, whose last element is the NULL pointer. These strings are filenames that are auxiliary to the main filename. The passed value may be NULL.

Returns:

A GDALDriverH handle or NULL on failure. For C++ applications this handle can be cast to a **GDALDriver** (p. **??**) *.

References GA_ReadOnly, GDALIdentifyDriver(), GDALDriverManager::GetDriver(), GDALDriverManager::GetDriverCount(), GDALDriver::pfnIdentify, and GDALDriver::pfnOpen.

Referenced by GDALCopyDatasetFiles(), GDALDeleteDataset(), GDALIdentifyDriver(), GDALRenameDataset(), and GDALDriver::QuietDelete().

44.11.4.82 int GDALInvGeoTransform (double * *gt_in*, double * *gt_out*)

Invert Geotransform.

This function will invert a standard 3x2 set of GeoTransform coefficients. This converts the equation from being pixel to geo to being geo to pixel.

Parameters:

gt_in Input geotransform (six doubles - unaltered).

gt_out Output geotransform (six doubles - updated).

Returns:

TRUE on success or FALSE if the equation is uninvertable.

References GDALInvGeoTransform().

Referenced by GDALCreateGenImgProjTransformer2(), GDALCreateRPCTransformer(), GDALInvGeoTransform(), and GDALSetGenImgProjTransformerDstGeoTransform().

44.11.4.83 int GDALLoadWorldFile (const char * *pszFilename*, double * *padfGeoTransform*)

Read ESRI world file.

This function reads an ESRI style world file, and formats a geotransform from its contents.

The world file contains an affine transformation with the parameters in a different order than in a geotransform array.

- geotransform[1] : width of pixel
- geotransform[4] : rotational coefficient, zero for north up images.
- geotransform[2] : rotational coefficient, zero for north up images.
- geotransform[5] : height of pixel (but negative)
- geotransform[0] + 0.5 * geotransform[1] + 0.5 * geotransform[2] : x offset to center of top left pixel.
- geotransform[3] + 0.5 * geotransform[4] + 0.5 * geotransform[5] : y offset to center of top left pixel.

Parameters:

pszFilename the world file name.

padfGeoTransform the six double array into which the geotransformation should be placed.

Returns:

TRUE on success or FALSE on failure.

References CPLAtofM(), and GDALLoadWorldFile().

Referenced by GDALLoadWorldFile(), and GDALReadWorldFile().

44.11.4.84 GDALDatasetH GDALOpen (const char * *pszFilename*, GDALAccess *eAccess*)

Open a raster file as a **GDALDataset** (p. **??**).

This function will try to open the passed file, or virtual dataset name by invoking the Open method of each registered **GDALDriver** (p. **??**) in turn. The first successful open will result in a returned dataset. If all drivers fail then NULL is returned.

Several recommandations :

- If you open a dataset object with GA_Update access, it is not recommanded to open a new dataset on the same underlying file.
- The returned dataset should only be accessed by one thread at a time. If you want to use it from different threads, you must add all necessary code (mutexes, etc.) to avoid concurrent use of the object. (Some drivers, such as GeoTIFF, maintain internal state variables that are updated each time a new block is read, thus preventing concurrent use.)

See also:

GDALOpenShared() (p. **??**)

Parameters:

pszFilename the name of the file to access. In the case of exotic drivers this may not refer to a physical file, but instead contain information for the driver on how to access a dataset.

eAccess the desired access, either GA_Update or GA_ReadOnly. Many drivers support only read only access.

Returns:

A GDALDatasetH handle or NULL on failure. For C++ applications this handle can be cast to a **GDALDataset** (p. **??**) *.

References GDALMajorObject::GetDescription(), GDALDriverManager::GetDriver(), GDALDriverManager::GetDriverCount(), GDALDriver::pfnOpen, GDALDataset::poDriver, and GDALMajorObject::SetDescription().

Referenced by GDALDriver::CopyFiles(), GDALDriver::Delete(), and GDALDriver::Rename().

44.11.4.85 GDALDatasetH GDALOpenShared (const char * *pszFilename*, GDALAccess *eAccess*)

Open a raster file as a **GDALDataset** (p. **??**).

This function works the same as **GDALOpen()** (p. **??**), but allows the sharing of **GDALDataset** (p. **??**) handles for a dataset with other callers to **GDALOpenShared()** (p. **??**).

In particular, **GDALOpenShared()** (p. **??**) will first consult it's list of currently open and shared GDALDataset's, and if the GetDescription() name for one exactly matches the pszFilename passed to **GDALOpenShared()** (p. **??**) it will be referenced and returned.

Starting with GDAL 1.6.0, if **GDALOpenShared()** (p. **??**) is called on the same pszFilename from two different threads, a different **GDALDataset** (p. **??**) object will be returned as it is not safe to use the same dataset from different threads, unless the user does explicitely use mutexes in its code.

See also:

GDALOpen() (p. **??**)

Parameters:

pszFilename the name of the file to access. In the case of exotic drivers this may not refer to a physical file, but instead contain information for the driver on how to access a dataset.

eAccess the desired access, either GA_Update or GA_ReadOnly. Many drivers support only read only access.

Returns:

A GDALDatasetH handle or NULL on failure. For C++ applications this handle can be cast to a **GDALDataset** (p. **??**) *.

References GA_ReadOnly, GA_Update, GDALDataset::GDALOpen, GDALMajorObject::GetDescription(), and GDALDataset::MarkAsShared().

44.11.4.86 double GDALPackedDMSToDec (double *dfPacked*)

Convert a packed DMS value (DDDMMMSSS.SS) into decimal degrees.

See **CPLPackedDMSToDec()** (p. **??**).

References GDALPackedDMSToDec().

Referenced by GDALPackedDMSToDec().

44.11.4.87 CPLErr GDALRasterIO (GDALRasterBandH *hBand*, GDALRWFlag *eRWFlag*, int *nXOff*, int *nYOff*, int *nXSize*, int *nYSize*, void * *pData*, int *nBufXSize*, int *nBufYSize*, GDALDataType *eBufType*, int *nPixelSpace*, int *nLineSpace*)

See also:

GDALRasterBand::RasterIO() (p. **??**)

References GDALRasterIO(), and GDALRasterBand::RasterIO().

Referenced by GDALChecksumImage(), GDALComputeMedianCutPCT(), GDALContourGenerate(), GDALDitherRGB2PCT(), GDALPolygonize(), GDALRasterIO(), GDALSieveFilter(), and GDALSimpleImageWarp().

44.11.4.88 CPLErr GDALReadBlock (GDALRasterBandH *hBand*, int *nXOff*, int *nYOff*, void * *pData*)

See also:

GDALRasterBand::ReadBlock() (p. **??**)

References GDALReadBlock(), and GDALRasterBand::ReadBlock().

Referenced by GDALReadBlock().

44.11.4.89 int GDALReadWorldFile (const char * *pszBaseFilename*, const char * *pszExtension*, double * *padfGeoTransform*)

Read ESRI world file.

This function reads an ESRI style world file, and formats a geotransform from it's contents. It does the same as **GDALLoadWorldFile()** (p. **??**) function, but it will form the filename for the worldfile from the filename of the raster file referred and the suggested extension. If no extension is provided, the code will internally try the unix style and windows style world file extensions (eg. for .tif these would be .tfw and .tifw).

The world file contains an affine transformation with the parameters in a different order than in a geotransform array.

- geotransform[1] : width of pixel
- geotransform[4] : rotational coefficient, zero for north up images.
- geotransform[2] : rotational coefficient, zero for north up images.
- geotransform[5] : height of pixel (but negative)
- geotransform[0] + 0.5 * geotransform[1] + 0.5 * geotransform[2] : x offset to center of top left pixel.
- geotransform[3] + 0.5 * geotransform[4] + 0.5 * geotransform[5] : y offset to center of top left pixel.

Parameters:

pszBaseFilename the target raster file.

pszExtension the extension to use (ie. ".wld") or NULL to derive it from the pszBaseFilename

padfGeoTransform the six double array into which the geotransformation should be placed.

Returns:

TRUE on success or FALSE on failure.

References CPLGetExtension(), CPLResetExtension(), GDALLoadWorldFile(), GDALReadWorldFile(), and VSIStatL().

Referenced by GDALReadWorldFile().

44.11.4.90 int GDALReferenceDataset (GDALDatasetH *hDataset*)

See also:

GDALDataset::Reference() (p. **??**)

References GDALReferenceDataset().

Referenced by GDALReferenceDataset().

44.11.4.91 CPLErr GDALRegenerateOverviews (GDALRasterBandH *hSrcBand*, int *nOverviewCount*, GDALRasterBandH * *pahOvrBands*, const char * *pszResampling*, GDALProgressFunc *pfnProgress*, void * *pProgressData*)

Generate downsampled overviews.

This function will generate one or more overview images from a base image using the requested downsampling algorithm. It's primary use is for generating overviews via **GDALDataset::BuildOverviews()** (p. **??**), but it can also be used to generate downsampled images in one file from another outside the overview architecture.

The output bands need to exist in advance.

The full set of resampling algorithms is documented in **GDALDataset::BuildOverviews()** (p. **??**).

Parameters:

hSrcBand the source (base level) band.

nOverviewCount the number of downsampled bands being generated.

pahOvrBands the list of downsampled bands to be generated.

pszResampling Resampling algorithm (eg. "AVERAGE").

pfnProgress progress report function.

pProgressData progress function callback data.

Returns:

CE_None on success or CE_Failure on failure.

References GDALRasterBand::FlushCache(), GCI_PaletteIndex, GDALDataTypeIsComplex(), GDALGetDataTypeSize(), GDALRegenerateOverviews(), GDT_Byte, GDT_CFloat32, GDT_Float32, GDALRasterBand::GetBand(), GDALRasterBand::GetBlockSize(), GDALRasterBand::GetColorInterpretation(), GDALRasterBand::GetColorTable(), GDALRasterBand::GetDataset(), GDALRasterBand::GetMaskBand(), GDALRasterBand::GetMaskFlags(), GDALMajorObject::GetMetadataItem(), GDALRasterBand::GetNoDataValue(), GDALColorTable::GetPaletteInterpretation(), GDALDataset::GetRasterCount(), GDALRasterBand::GetRasterDataType(), GDALRasterBand::GetXSize(), GDALRasterBand::GetYSize(), GF_Read, GPI_RGB, GDALRasterBand::RasterIO(), VSIMalloc2(), and VSIMalloc3().

Referenced by GDALRegenerateOverviews().

44.11.4.92 int GDALRegisterDriver (GDALDriverH *hDriver*)

See also:

GDALDriverManager::GetRegisterDriver()

References GDALRegisterDriver(), and GDALDriverManager::RegisterDriver().

Referenced by GDALRegisterDriver().

44.11.4.93 CPLErr GDALRenameDataset (GDALDriverH *hDriver*, const char * *pszNewName*, const char * *pszOldName*)

See also:

GDALDriver::Rename() (p. **??**)

References GDALIdentifyDriver(), and GDALRenameDataset().

Referenced by GDALRenameDataset().

44.11.4.94 int GDALScaledProgress (double *dfComplete*, const char * *pszMessage*, void * *pData*)

Scaled progress transformer.

This is the progress function that should be passed along with the callback data returned by **GDALCreateScaledProgress()** (p. **??**).

References GDALScaledProgress().

Referenced by GDALScaledProgress().

44.11.4.95 void GDALSetCacheMax (int *nNewSize*)

Set maximum cache memory.

This function sets the maximum amount of memory that GDAL is permitted to use for GDALRasterBlock caching.

Parameters:

nNewSize the maximum number of bytes for caching. Maximum is 2GB.

References GDALSetCacheMax().

Referenced by GDALSetCacheMax().

44.11.4.96 void GDALSetDescription (GDALMajorObjectH *hObject*, const char * *pszNewDesc*)

See also:

GDALMajorObject::SetDescription() (p. **??**)

References GDALSetDescription().

Referenced by GDALSetDescription().

44.11.4.97 CPLErr GDALSetGCPs (GDALDatasetH *hDS*, int *nGCPCount*, const GDAL_GCP * *pasGCPList*, const char * *pszGCPProjection*)

See also:

GDALDataset::SetGCPs() (p. **??**)

References GDALSetGCPs().

Referenced by GDALSetGCPs().

44.11.4.98 CPLErr GDALSetGeoTransform (GDALDatasetH *hDS*, double * *padfTransform*)

See also:

GDALDataset::SetGeoTransform() (p. **??**)

References GDALSetGeoTransform(), and GDALDataset::SetGeoTransform().

Referenced by GDALSetGeoTransform().

44.11.4.99 CPLErr GDALSetMetadata (GDALMajorObjectH *hObject*, char ** *papszMD*, const char * *pszDomain*)

See also:

GDALMajorObject::SetMetadata() (p. **??**)

References GDALSetMetadata().

Referenced by GDALSetMetadata().

44.11.4.100 CPLErr GDALSetMetadataItem (GDALMajorObjectH *hObject*, const char * *pszName*, const char * *pszValue*, const char * *pszDomain*)

See also:

GDALMajorObject::SetMetadataItem() (p. **??**)

References GDALSetMetadataItem().

Referenced by GDALSetMetadataItem().

44.11.4.101 CPLErr GDALSetProjection (GDALDatasetH *hDS*, const char * *pszProjection*)

See also:

GDALDataset::SetProjection() (p. **??**)

References GDALSetProjection(), and GDALDataset::SetProjection().

Referenced by GDALAutoCreateWarpedVRT(), and GDALSetProjection().

44.11.4.102 CPLErr GDALSetRasterCategoryNames (GDALRasterBandH *hBand*, char ** *papszNames*)

See also:

GDALRasterBand::SetCategoryNames() (p. **??**)

References GDALSetRasterCategoryNames().

Referenced by GDALSetRasterCategoryNames().

44.11.4.103 CPLErr GDALSetRasterColorInterpretation (GDALRasterBandH *hBand*, GDALColorInterp *eColorInterp*)

See also:

GDALRasterBand::SetColorInterpretation() (p. **??**)

References GDALSetRasterColorInterpretation().

Referenced by GDALSetRasterColorInterpretation().

44.11.4.104 CPLErr GDALSetRasterColorTable (GDALRasterBandH *hBand*, GDALColorTableH *hCT*)

See also:

GDALRasterBand::SetColorTable() (p. **??**)

References GDALSetRasterColorTable().

Referenced by GDALSetRasterColorTable().

44.11.4.105 CPLErr GDALSetRasterNoDataValue (GDALRasterBandH *hBand*, double *dfValue*)

See also:

GDALRasterBand::SetNoDataValue() (p. **??**)

References GDALSetRasterNoDataValue().

Referenced by GDALSetRasterNoDataValue().

44.11.4.106 void GDALSwapWords (void * *pData*, int *nWordSize*, int *nWordCount*, int *nWordSkip*)

Byte swap words in-place.

This function will byte swap a set of 2, 4 or 8 byte words "in place" in a memory array. No assumption is made that the words being swapped are word aligned in memory. Use the CPL_LSB and CPL_MSB macros from **cpl_port.h** (p. **??**) to determine if the current platform is big endian or little endian. Use The macros like CPL_SWAP32() to byte swap single values without the overhead of a function call.

Parameters:

pData pointer to start of data buffer.

nWordSize size of words being swapped in bytes. Normally 2, 4 or 8.

nWordCount the number of words to be swapped in this call.

nWordSkip the byte offset from the start of one word to the start of the next. For packed buffers this is the same as nWordSize.

References GDALSwapWords().

Referenced by GDALSwapWords().

44.11.4.107 int GDALTermProgress (double *dfComplete*, const char * *pszMessage*, void * *pProgressArg*)

Simple progress report to terminal.

This progress reporter prints simple progress report to the terminal window. The progress report generally looks something like this:

```
0...10...20...30...40...50...60...70...80...90...100 - done.
```

Every 2.5% of progress another number or period is emitted. Note that **GDALTermProgress()** (p. **??**) uses internal static data to keep track of the last percentage reported and will get confused if two terminal based progress reportings are active at the same time.

The **GDALTermProgress()** (p. **??**) function maintains an internal memory of the last percentage complete reported in a static variable, and this makes it unsuitable to have multiple **GDALTermProgress()** (p. **??**)'s active eithin a single thread or across multiple threads.

Parameters:

dfComplete completion ratio from 0.0 to 1.0.

pszMessage optional message.

pProgressArg ignored callback data argument.

Returns:

Always returns TRUE indicating the process should continue.

References GDALTermProgress().

Referenced by GDALTermProgress().

44.11.4.108 int GDALValidateCreationOptions (GDALDriverH *hDriver*, char ** *papszCreationOptions*)

Validate the list of creation options that are handled by a driver

This is a helper method primarily used by Create() and CreateCopy() to validate that the passed in list of creation options is compatible with the GDAL_DMD_CREATIONOPTIONLIST metadata item defined by some drivers.

See also:

GDALGetDriverCreationOptionList() (p. **??**)

If the GDAL_DMD_CREATIONOPTIONLIST metadata item is not defined, this function will return TRUE. Otherwise it will check that the keys and values in the list of creation options are compatible with the capabilities declared by the GDAL_DMD_CREATIONOPTIONLIST metadata item. In case of incompatibility a (non fatal) warning will be emited and FALSE will be returned.

Parameters:

hDriver the handle of the driver with whom the lists of creation option must be validated

papszCreationOptions the list of creation options. An array of strings, whose last element is a NULL pointer

Returns:

TRUE if the list of creation options is compatible with the Create() and CreateCopy() method of the driver, FALSE otherwise.

References CPLStrtod(), GDALValidateCreationOptions(), GDALMajorObject::GetDescription(), CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

Referenced by GDALDriver::Create(), GDALDriver::CreateCopy(), and GDALValidateCreationOptions().

44.11.4.109 const char* GDALVersionInfo (const char * *pszRequest*)

Get runtime version information.

Available pszRequest values:

- "VERSION_NUM": Returns GDAL_VERSION_NUM formatted as a string. ie. "1170"
- "RELEASE_DATE": Returns GDAL_RELEASE_DATE formatted as a string. ie. "20020416".
- "RELEASE_NAME": Returns the GDAL_RELEASE_NAME. ie. "1.1.7"
- "–version": Returns one line version message suitable for use in response to –version requests. ie. "GDAL 1.1.7, released 2002/04/16"
- "LICENCE": Returns the content of the LICENSE.TXT file from the GDAL_DATA directory. Currently, the returned string is leaking memory but applications are discouraged from deallocating the returned memory in this case since in the future we may resolve the leak issue internally.

Parameters:

pszRequest the type of version info desired, as listed above.

Returns:

an internal string containing the requested information.

References GDALVersionInfo(), VSIFCloseL(), VSIFOpenL(), VSIFReadL(), VSIFSeekL(), and VSIFTellL().

Referenced by GDALGeneralCmdLineProcessor(), and GDALVersionInfo().

44.11.4.110 CPLErr GDALWriteBlock (GDALRasterBandH *hBand*, int *nXOff*, int *nYOff*, void * *pData*)

See also:

GDALRasterBand::WriteBlock() (p. **??**)

References GDALWriteBlock(), and GDALRasterBand::WriteBlock().

Referenced by GDALWriteBlock().

44.11.4.111 int GDALWriteWorldFile (const char * *pszBaseFilename*, const char * *pszExtension*, double * *padfGeoTransform*)

Write ESRI world file.

This function writes an ESRI style world file from the passed geotransform.

The world file contains an affine transformation with the parameters in a different order than in a geotransform array.

- geotransform[1] : width of pixel
- geotransform[4] : rotational coefficient, zero for north up images.
- geotransform[2] : rotational coefficient, zero for north up images.
- geotransform[5] : height of pixel (but negative)
- geotransform[0] + 0.5 * geotransform[1] + 0.5 * geotransform[2] : x offset to center of top left pixel.
- geotransform[3] + 0.5 * geotransform[4] + 0.5 * geotransform[5] : y offset to center of top left pixel.

Parameters:

pszBaseFilename the target raster file.

pszExtension the extension to use (ie. ".wld"). Must not be NULL

padfGeoTransform the six double array from which the geotransformation should be read.

Returns:

TRUE on success or FALSE on failure.

References CPLResetExtension(), GDALWriteWorldFile(), VSIFCloseL(), VSIFOpenL(), and VSIFWriteL().

Referenced by GDALWriteWorldFile().

44.12 gdal_alg.h File Reference

```
#include "gdal.h"
#include "cpl_minixml.h"
#include "ogr_api.h"
```

Classes

- struct **GDALTransformerInfo**
- struct **OGRContourWriterInfo**
- struct **GDALGridInverseDistanceToAPowerOptions**
- struct **GDALGridMovingAverageOptions**
- struct **GDALGridNearestNeighborOptions**
- struct **GDALGridDataMetricsOptions**

Typedefs

- typedef int(∗ **GDALTransformerFunc**)(void ∗pTransformerArg, int bDstToSrc, int nPointCount, double ∗x, double ∗y, double ∗z, int ∗panSuccess)
- typedef CPLErr(∗ **GDALContourWriter**)(double dfLevel, int nPoints, double ∗padfX, double ∗padfY, void ∗)
- typedef void ∗ **GDALContourGeneratorH**

Enumerations

- enum **GDALGridAlgorithm** {

  GGA_InverseDistanceToAPower = 1, **GGA_MovingAverage** = 2, **GGA_NearestNeighbor** = 3, **GGA_MetricMinimum** = 4,

  GGA_MetricMaximum = 5, **GGA_MetricRange** = 6 }

Functions

- int **GDALComputeMedianCutPCT** (GDALRasterBandH hRed, GDALRasterBandH hGreen, GDALRasterBandH hBlue, int(∗pfnIncludePixel)(int, int, void ∗), int nColors, GDALColorTableH hColorTable, GDALProgressFunc pfnProgress, void ∗pProgressArg)
- int **GDALDitherRGB2PCT** (GDALRasterBandH hRed, GDALRasterBandH hGreen, GDALRasterBandH hBlue, GDALRasterBandH hTarget, GDALColorTableH hColorTable, GDALProgressFunc pfnProgress, void ∗pProgressArg)
- int **GDALChecksumImage** (GDALRasterBandH hBand, int nXOff, int nYOff, int nXSize, int nYSize)
- CPLErr **GDALComputeProximity** (GDALRasterBandH hSrcBand, GDALRasterBandH hProximityBand, char ∗∗papszOptions, GDALProgressFunc pfnProgress, void ∗pProgressArg)
- CPLErr **GDALFillNodata** (GDALRasterBandH hTargetBand, GDALRasterBandH hMaskBand, double dfMaxSearchDist, int bConicSearch, int nSmoothingIterations, char ∗∗papszOptions, GDALProgressFunc pfnProgress, void ∗pProgressArg)
- CPLErr **GDALPolygonize** (GDALRasterBandH hSrcBand, GDALRasterBandH hMaskBand, OGRLayerH hOutLayer, int iPixValField, char ∗∗papszOptions, GDALProgressFunc pfnProgress, void ∗pProgressArg)

- CPLErr **GDALSieveFilter** (GDALRasterBandH hSrcBand, GDALRasterBandH hMaskBand, GDALRasterBandH hDstBand, int nSizeThreshold, int nConnectedness, char **papszOptions, GDALProgressFunc pfnProgress, void *pProgressArg)
- void **GDALDestroyTransformer** (void *pTransformerArg)
- int **GDALUseTransformer** (void *pTranformerArg, int bDstToSrc, int nPointCount, double *x, double *y, double *z, int *panSuccess)
- void * **GDALCreateGenImgProjTransformer** (GDALDatasetH hSrcDS, const char *pszSrcWKT, GDALDatasetH hDstDS, const char *pszDstWKT, int bGCPUseOK, double dfGCPErrorThreshold, int nOrder)
- void * **GDALCreateGenImgProjTransformer2** (GDALDatasetH hSrcDS, GDALDatasetH hDstDS, char **papszOptions)
- void **GDALSetGenImgProjTransformerDstGeoTransform** (void *, const double *)
- void **GDALDestroyGenImgProjTransformer** (void *)
- int **GDALGenImgProjTransform** (void *pTransformArg, int bDstToSrc, int nPointCount, double *x, double *y, double *z, int *panSuccess)
- void * **GDALCreateReprojectionTransformer** (const char *pszSrcWKT, const char *pszDstWKT)
- void **GDALDestroyReprojectionTransformer** (void *)
- int **GDALReprojectionTransform** (void *pTransformArg, int bDstToSrc, int nPointCount, double *x, double *y, double *z, int *panSuccess)
- void * **GDALCreateGCPTransformer** (int nGCPCount, const **GDAL_GCP** *pasGCPList, int nReqOrder, int bReversed)
- void **GDALDestroyGCPTransformer** (void *pTransformArg)
- int **GDALGCPTransform** (void *pTransformArg, int bDstToSrc, int nPointCount, double *x, double *y, double *z, int *panSuccess)
- void * **GDALCreateTPSTransformer** (int nGCPCount, const **GDAL_GCP** *pasGCPList, int bReversed)
- void **GDALDestroyTPSTransformer** (void *pTransformArg)
- int **GDALTPSTransform** (void *pTransformArg, int bDstToSrc, int nPointCount, double *x, double *y, double *z, int *panSuccess)
- void * **GDALCreateRPCTransformer** (GDALRPCInfo *psRPC, int bReversed, double dfPixErrThreshold, char **papszOptions)
- void **GDALDestroyRPCTransformer** (void *pTransformArg)
- int **GDALRPCTransform** (void *pTransformArg, int bDstToSrc, int nPointCount, double *x, double *y, double *z, int *panSuccess)
- void * **GDALCreateGeoLocTransformer** (GDALDatasetH hBaseDS, char **papszGeolocationInfo, int bReversed)
- void **GDALDestroyGeoLocTransformer** (void *pTransformArg)
- int **GDALGeoLocTransform** (void *pTransformArg, int bDstToSrc, int nPointCount, double *x, double *y, double *z, int *panSuccess)
- void * **GDALCreateApproxTransformer** (**GDALTransformerFunc** pfnRawTransformer, void *pRawTransformerArg, double dfMaxError)
- void **GDALApproxTransformerOwnsSubtransformer** (void *pCBData, int bOwnFlag)
- void **GDALDestroyApproxTransformer** (void *pApproxArg)
- int **GDALApproxTransform** (void *pTransformArg, int bDstToSrc, int nPointCount, double *x, double *y, double *z, int *panSuccess)
- int **GDALSimpleImageWarp** (GDALDatasetH hSrcDS, GDALDatasetH hDstDS, int nBandCount, int *panBandList, **GDALTransformerFunc** pfnTransform, void *pTransformArg, GDALProgressFunc pfnProgress, void *pProgressArg, char **papszWarpOptions)
- CPLErr **GDALSuggestedWarpOutput** (GDALDatasetH hSrcDS, **GDALTransformerFunc** pfnTransformer, void *pTransformArg, double *padfGeoTransformOut, int *pnPixels, int *pnLines)

- CPLErr **GDALSuggestedWarpOutput2** (GDALDatasetH hSrcDS, **GDALTransformerFunc** pfnTransformer, void *pTransformArg, double *padfGeoTransformOut, int *pnPixels, int *pnLines, double *padfExtents, int nOptions)
- **CPLXMLNode** * **GDALSerializeTransformer** (**GDALTransformerFunc** pfnFunc, void *pTransformArg)
- CPLErr **GDALDeserializeTransformer** (**CPLXMLNode** *psTree, **GDALTransformerFunc** *ppfnFunc, void **ppTransformArg)
- GDALContourGeneratorH **GDAL_CG_Create** (int nWidth, int nHeight, int bNoDataSet, double dfNoDataValue, double dfContourInterval, double dfContourBase, GDALContourWriter pfnWriter, void *pCBData)
- CPLErr **GDAL_CG_FeedLine** (GDALContourGeneratorH hCG, double *padfScanline)
- void **GDAL_CG_Destroy** (GDALContourGeneratorH hCG)
- CPLErr **OGRContourWriter** (double, int, double *, double *, void *pInfo)
- CPLErr **GDALContourGenerate** (GDALRasterBandH hBand, double dfContourInterval, double dfContourBase, int nFixedLevelCount, double *padfFixedLevels, int bUseNoData, double dfNoDataValue, void *hLayer, int iIDField, int iElevField, GDALProgressFunc pfnProgress, void *pProgressArg)
- CPLErr **GDALRasterizeGeometries** (GDALDatasetH hDS, int nBandCount, int *panBandList, int nGeomCount, OGRGeometryH *pahGeometries, **GDALTransformerFunc** pfnTransformer, void *pTransformArg, double *padfGeomBurnValue, char **papszOptions, GDALProgressFunc pfnProgress, void *pProgressArg)
- CPLErr **GDALRasterizeLayers** (GDALDatasetH hDS, int nBandCount, int *panBandList, int nLayerCount, OGRLayerH *pahLayers, **GDALTransformerFunc** pfnTransformer, void *pTransformArg, double *padfLayerBurnValues, char **papszOptions, GDALProgressFunc pfnProgress, void *pProgressArg)
- CPLErr **GDALGridCreate** (**GDALGridAlgorithm**, const void *, GUInt32, const double *, const double *, const double *, double, double, double, double, GUInt32, GUInt32, **GDALDataType**, void *, GDALProgressFunc, void *)

44.12.1 Detailed Description

Public (C callable) GDAL algorithm entry points, and definitions.

44.12.2 Typedef Documentation

44.12.2.1 int GDALTransformerFunc

Generic signature for spatial point transformers.

This function signature is used for a variety of functions that accept passed in functions used to transform point locations between two coordinate spaces.

The **GDALCreateGenImgProjTransformer()** (p. **??**), **GDALCreateReprojectionTransformer()** (p. **??**), **GDALCreateGCPTransformer()** (p. **??**) and **GDALCreateApproxTransformer()** (p. **??**) functions can be used to prepare argument data for some built-in transformers. As well, applications can implement their own transformers to the following signature.

```
typedef int
(*GDALTransformerFunc)( void *pTransformerArg,
                        int bDstToSrc, int nPointCount,
                        double *x, double *y, double *z, int *panSuccess );
```

Parameters:

pTransformerArg application supplied callback data used by the transformer.

bDstToSrc if TRUE the transformation will be from the destination coordinate space to the source coordinate system, otherwise the transformation will be from the source coordinate system to the destination coordinate system.

nPointCount number of points in the x, y and z arrays.

x input X coordinates. Results returned in same array.

y input Y coordinates. Results returned in same array.

z input Z coordinates. Results returned in same array.

panSuccess array of ints in which success (TRUE) or failure (FALSE) flags are returned for the translation of each point.

Returns:

TRUE if the overall transformation succeeds (though some individual points may have failed) or FALSE if the overall transformation fails.

44.12.3 Enumeration Type Documentation

44.12.3.1 enum GDALGridAlgorithm

Gridding Algorithms

Enumerator:

GGA_InverseDistanceToAPower Inverse distance to a power

GGA_MovingAverage Moving Average

GGA_NearestNeighbor Nearest Neighbor

GGA_MetricMinimum Minimum Value (Data Metric)

GGA_MetricMaximum Maximum Value (Data Metric)

GGA_MetricRange Data Range (Data Metric)

44.12.4 Function Documentation

44.12.4.1 int GDALApproxTransform (void * *pCBData*, int *bDstToSrc*, int *nPoints*, double * *x*, double * *y*, double * *z*, int * *panSuccess*)

Perform approximate transformation.

Actually performs the approximate transformation described in **GDALCreateApproxTransformer()** (p. **??**). This function matches the **GDALTransformerFunc()** (p. **??**) signature. Details of the arguments are described there.

References GDALApproxTransform().

Referenced by GDALApproxTransform(), GDALAutoCreateWarpedVRT(), GDALCreateApproxTransformer(), and GDALReprojectImage().

44.12.4.2 int GDALChecksumImage (GDALRasterBandH *hBand*, int *nXOff*, int *nYOff*, int *nXSize*, int *nYSize*)

Compute checksum for image region.

Computes a 16bit (0-65535) checksum from a region of raster data on a GDAL supported band. Floating point data is converted to 32bit integer so decimal portions of such raster data will not affect the checksum. Real and Imaginary components of complex bands influence the result.

Parameters:

hBand the raster band to read from.

nXOff pixel offset of window to read.

nYOff line offset of window to read.

nXSize pixel size of window to read.

nYSize line size of window to read.

Returns:

Checksum value.

References GDALChecksumImage(), GDALDataTypeIsComplex(), GDALGetRasterDataType(), GDALRasterIO(), GDT_CFloat32, GDT_CFloat64, GDT_CInt32, GDT_Float32, GDT_Float64, GDT_Int32, GF_Read, and VSIMalloc2().

Referenced by GDALChecksumImage().

44.12.4.3 int GDALComputeMedianCutPCT (GDALRasterBandH *hRed*, GDALRasterBandH *hGreen*, GDALRasterBandH *hBlue*, int(∗)(int, int, void ∗) *pfnIncludePixel*, int *nColors*, GDALColorTableH *hColorTable*, GDALProgressFunc *pfnProgress*, void ∗ *pProgressArg*)

Compute optimal PCT for RGB image.

This function implements a median cut algorithm to compute an "optimal" pseudocolor table for representing an input RGB image. This PCT could then be used with **GDALDitherRGB2PCT()** (p. **??**) to convert a 24bit RGB image into an eightbit pseudo-colored image.

This code was based on the tiffmedian.c code from libtiff (www.libtiff.org) which was based on a paper by Paul Heckbert:

```
*   "Color  Image Quantization for Frame Buffer Display", Paul
*   Heckbert, SIGGRAPH proceedings, 1982, pp. 297-307.
*
```

The red, green and blue input bands do not necessarily need to come from the same file, but they must be the same width and height. They will be clipped to 8bit during reading, so non-eight bit bands are generally inappropriate.

Parameters:

hRed Red input band.

hGreen Green input band.

hBlue Blue input band.

pfnIncludePixel function used to test which pixels should be included in the analysis. At this time this argument is ignored and all pixels are utilized. This should normally be NULL.

nColors the desired number of colors to be returned (2-256).

hColorTable the colors will be returned in this color table object.

pfnProgress callback for reporting algorithm progress matching the GDALProgressFunc() semantics. May be NULL.

pProgressArg callback argument passed to pfnProgress.

Returns:

returns CE_None on success or CE_Failure if an error occurs.

References GDALColorEntry::c1, GDALColorEntry::c2, GDALColorEntry::c3, GDALColorEntry::c4, GDALComputeMedianCutPCT(), GDALGetRasterBandXSize(), GDALGetRasterBandYSize(), GDALRasterIO(), GDT_Byte, and GF_Read.

Referenced by GDALComputeMedianCutPCT().

44.12.4.4 CPLErr GDALContourGenerate (GDALRasterBandH *hBand*, double *dfContourInterval*, double *dfContourBase*, int *nFixedLevelCount*, double * *padfFixedLevels*, int *bUseNoData*, double *dfNoDataValue*, void * *hLayer*, int *iIDField*, int *iElevField*, GDALProgressFunc *pfnProgress*, void * *pProgressArg*)

Create vector contours from raster DEM.

This algorithm will generate contours vectors for the input raster band on the requested set of contour levels. The vector contours are written to the passed in OGR vector layer. Also, a NODATA value may be specified to identify pixels that should not be considered in contour line generation.

The gdal/apps/gdal_contour.cpp mainline can be used as an example of how to use this function.

ALGORITHM RULES

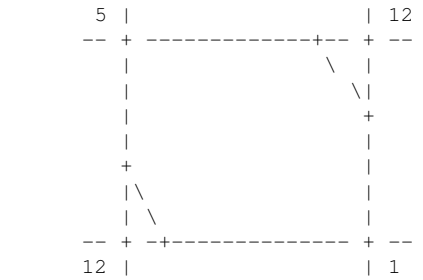
For contouring purposes raster pixel values are assumed to represent a point value at the center of the corresponding pixel region. For the purpose of contour generation we virtually connect each pixel center to the values to the left, right, top and bottom. We assume that the pixel value is linearly interpolated between the pixel centers along each line, and determine where (if any) contour lines will appear onlong these line segements. Then the contour crossings are connected.

This means that contour lines nodes won't actually be on pixel edges, but rather along vertical and horizontal lines connecting the pixel centers.

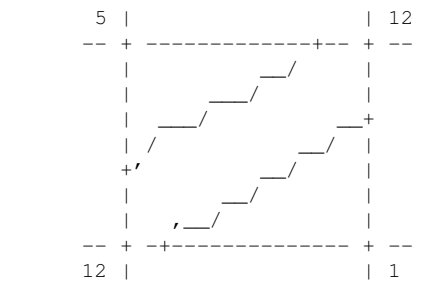
```
General Case:

      5 |                  | 3
     -- + ---------------- + --
        |                  |
        |                  |
        |                  |
        |                  |
     10 +                  |
        |\                 |
        | \                |
     -- + -+--------------- + --
     12 |  10              | 1


Saddle Point:
```

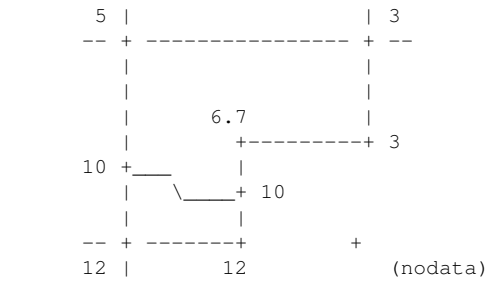


or:



Nodata:

In the "nodata" case we treat the whole nodata pixel as a no-mans land. We extend the corner pixels near the nodata out to half way and then construct extra lines from those points to the center which is assigned an averaged value from the two nearby points (in this case (12+3+5)/3).



Parameters:

hBand The band to read raster data from. The whole band will be processed.

dfContourInterval The elevation interval between contours generated.

dfContourBase The "base" relative to which contour intervals are applied. This is normally zero, but could be different. To generate 10m contours at 5, 15, 25, ... the ContourBase would be 5.

nFixedLevelCount The number of fixed levels. If this is greater than zero, then fixed levels will be used, and ContourInterval and ContourBase are ignored.

padfFixedLevels The list of fixed contour levels at which contours should be generated. It will contain FixedLevelCount entries, and may be NULL if fixed levels are disabled (FixedLevelCount = 0).

bUseNoData If TRUE the dfNoDataValue will be used.

dfNoDataValue the value to use as a "nodata" value. That is, a pixel value which should be ignored in generating contours as if the value of the pixel were not known.

hLayer the layer to which new contour vectors will be written. Each contour will have a LINESTRING geometry attached to it. This is really of type OGRLayerH, but void * is used to avoid pulling the ogr_api.h file in here.

iIDField if not -1 this will be used as a field index to indicate where a unique id should be written for each feature (contour) written.

iElevField if not -1 this will be used as a field index to indicate where the elevation value of the contour should be written.

pfnProgress a GDALProgressFunc that may be used to report progress to the user, or to interrupt the algorithm. May be NULL if not required.

pProgressArg the callback data for the pfnProgress function.

Returns:

CE_None on success or CE_Failure if an error occurs.

References GDALGetBandDataset(), GDALGetGeoTransform(), GDALGetRasterBandXSize(), GDALGetRasterBandYSize(), GDALRasterIO(), GDT_Float64, and GF_Read.

44.12.4.5 void* GDALCreateApproxTransformer (GDALTransformerFunc *pfnBaseTransformer*, void * *pBaseTransformArg*, double *dfMaxError*)

Create an approximating transformer.

This function creates a context for an approximated transformer. Basically a high precision transformer is supplied as input and internally linear approximations are computed to generate results to within a defined precision.

The approximation is actually done at the point where **GDALApproxTransform()** (p. **??**) calls are made, and depend on the assumption that the roughly linear. The first and last point passed in must be the extreme values and the intermediate values should describe a curve between the end points. The approximator transforms and center using the approximate transformer, and then compares the true middle transformed value to a linear approximation based on the end points. If the error is within the supplied threshold then the end points are used to linearly approximate all the values otherwise the inputs points are split into two smaller sets, and the function recursively called till a sufficiently small set of points if found that the linear approximation is OK, or that all the points are exactly computed.

This function is very suitable for approximating transformation results from output pixel/line space to input coordinates for warpers that operate on one input scanline at a time. Care should be taken using it in other circumstances as little internal validation is done, in order to keep things fast.

Parameters:

pfnBaseTransformer the high precision transformer which should be approximated.

pBaseTransformArg the callback argument for the high precision transformer.

dfMaxError the maximum cartesian error in the "output" space that is to be accepted in the linear approximation.

Returns:

callback pointer suitable for use with **GDALApproxTransform()** (p. **??**). It should be deallocated with **GDALDestroyApproxTransformer()** (p. **??**).

References GDALApproxTransform(), GDALCreateApproxTransformer(), and GDALDestroyApproxTransformer().

Referenced by GDALAutoCreateWarpedVRT(), GDALCreateApproxTransformer(), and GDALReprojectImage().

44.12.4.6 void* GDALCreateGCPTransformer (int *nGCPCount*, const GDAL_GCP * *pasGCPList*, int *nReqOrder*, int *bReversed*)

Create GCP based polynomial transformer.

Computes least squares fit polynomials from a provided set of GCPs, and stores the coefficients for later transformation of points between pixel/line and georeferenced coordinates.

The return value should be used as a TransformArg in combination with the transformation function GDALGCPTransform which fits the GDALTransformerFunc signature. The returned transform argument should be deallocated with GDALDestroyGCPTransformer when no longer needed.

This function may fail (returning NULL) if the provided set of GCPs are inadequate for the requested order, the determinate is zero or they are otherwise "ill conditioned".

Note that 2nd order requires at least 6 GCPs, and 3rd order requires at least 10 gcps. If nReqOrder is 0 the highest order possible with the provided gcp count will be used.

Parameters:

nGCPCount the number of GCPs in pasGCPList.

pasGCPList an array of GCPs to be used as input.

nReqOrder the requested polynomial order. It should be 1, 2 or 3.

Returns:

the transform argument or NULL if creation fails.

References GDAL_GCP::dfGCPLine, GDAL_GCP::dfGCPPixel, GDAL_GCP::dfGCPX, GDAL_GCP::dfGCPY, GDALCreateGCPTransformer(), GDALDestroyGCPTransformer(), and GDALGCPTransform().

Referenced by GDALCreateGCPTransformer(), and GDALCreateGenImgProjTransformer2().

44.12.4.7 void* GDALCreateGenImgProjTransformer (GDALDatasetH *hSrcDS*, const char * *pszSrcWKT*, GDALDatasetH *hDstDS*, const char * *pszDstWKT*, int *bGCPUseOK*, double *dfGCPErrorThreshold*, int *nOrder*)

Create image to image transformer.

This function creates a transformation object that maps from pixel/line coordinates on one image to pixel/line coordinates on another image. The images may potentially be georeferenced in different coordinate systems, and may used GCPs to map between their pixel/line coordinates and georeferenced coordinates (as opposed to the default assumption that their geotransform should be used).

This transformer potentially performs three concatenated transformations.

The first stage is from source image pixel/line coordinates to source image georeferenced coordinates, and may be done using the geotransform, or if not defined using a polynomial model derived from GCPs. If GCPs are used this stage is accomplished using **GDALGCPTransform()** (p. **??**).

The second stage is to change projections from the source coordinate system to the destination coordinate system, assuming they differ. This is accomplished internally using **GDALReprojectionTransform()** (p. **??**).

The third stage is converting from destination image georeferenced coordinates to destination image coordinates. This is done using the destination image geotransform, or if not available, using a polynomial model derived from GCPs. If GCPs are used this stage is accomplished using **GDALGCPTransform()** (p. **??**). This stage is skipped if hDstDS is NULL when the transformation is created.

Parameters:

hSrcDS source dataset, or NULL.

pszSrcWKT the coordinate system for the source dataset. If NULL, it will be read from the dataset itself.

hDstDS destination dataset (or NULL).

pszDstWKT the coordinate system for the destination dataset. If NULL, and hDstDS not NULL, it will be read from the destination dataset.

bGCPUseOK TRUE if GCPs should be used if the geotransform is not available on the source dataset (not destination).

dfGCPErrorThreshold ignored/deprecated.

nOrder the maximum order to use for GCP derived polynomials if possible. Use 0 to autoselect, or -1 for thin plate splines.

Returns:

handle suitable for use **GDALGenImgProjTransform()** (p. **??**), and to be deallocated with **GDALDestroyGenImgProjTransformer()** (p. **??**).

References GDALCreateGenImgProjTransformer(), and GDALCreateGenImgProjTransformer2().

Referenced by GDALAutoCreateWarpedVRT(), GDALCreateGenImgProjTransformer(), GDALRasterizeGeometries(), and GDALReprojectImage().

44.12.4.8 void* GDALCreateGenImgProjTransformer2 (GDALDatasetH *hSrcDS*, GDALDatasetH *hDstDS*, char ** *papszOptions*)

Create image to image transformer.

This function creates a transformation object that maps from pixel/line coordinates on one image to pixel/line coordinates on another image. The images may potentially be georeferenced in different coordinate systems, and may used GCPs to map between their pixel/line coordinates and georeferenced coordinates (as opposed to the default assumption that their geotransform should be used).

This transformer potentially performs three concatenated transformations.

The first stage is from source image pixel/line coordinates to source image georeferenced coordinates, and may be done using the geotransform, or if not defined using a polynomial model derived from GCPs. If GCPs are used this stage is accomplished using **GDALGCPTransform()** (p. **??**).

The second stage is to change projections from the source coordinate system to the destination coordinate system, assuming they differ. This is accomplished internally using **GDALReprojectionTransform()** (p. **??**).

The third stage is converting from destination image georeferenced coordinates to destination image coordinates. This is done using the destination image geotransform, or if not available, using a polynomial model derived from GCPs. If GCPs are used this stage is accomplished using **GDALGCPTransform()** (p. **??**). This stage is skipped if hDstDS is NULL when the transformation is created.

Supported Options:

- SRC_SRS: WKT SRS to be used as an override for hSrcDS.
- DST_SRS: WKT SRS to be used as an override for hDstDS.
- GCPS_OK: If false, GCPs will not be used, default is TRUE.
- MAX_GCP_ORDER: the maximum order to use for GCP derived polynomials if possible. The default is to autoselect based on the number of GCPs. A value of -1 triggers use of Thin Plate Spline instead of polynomials.
- METHOD: may have a value which is one of GEOTRANSFORM, GCP_POLYNOMIAL, GCP_TPS, GEOLOC_ARRAY, RPC to force only one geolocation method to be considered on the source dataset.
- RPC_HEIGHT: A fixed height to be used with RPC calculations.
- RPC_DEM: The name of a DEM file to be used with RPC calculations.

Parameters:

hSrcDS source dataset, or NULL.

hDstDS destination dataset (or NULL).

Returns:

handle suitable for use **GDALGenImgProjTransform()** (p. **??**), and to be deallocated with **GDALDestroyGenImgProjTransformer()** (p. **??**) or NULL on failure.

References GDALCreateGCPTransformer(), GDALCreateGenImgProjTransformer2(), GDALCreateReprojectionTransformer(), GDALCreateRPCTransformer(), GDALCreateTPSTransformer(), GDALDestroyGenImgProjTransformer(), GDALGenImgProjTransform(), GDALGetDescription(), GDALGetGCPCount(), GDALGetGCPProjection(), GDALGetGCPs(), GDALGetGeoTransform(), GDALGetMetadata(), GDALGetProjectionRef(), and GDALInvGeoTransform().

Referenced by GDALCreateGenImgProjTransformer(), and GDALCreateGenImgProjTransformer2().

44.12.4.9 void* GDALCreateReprojectionTransformer (const char * *pszSrcWKT*, const char * *pszDstWKT*)

Create reprojection transformer.

Creates a callback data structure suitable for use with GDALReprojectionTransformation() to represent a transformation from one geographic or projected coordinate system to another. On input the coordinate systems are described in OpenGIS WKT format.

Internally the OGRCoordinateTransformation object is used to implement the reprojection.

Parameters:

pszSrcWKT the coordinate system for the source coordinate system.

pszDstWKT the coordinate system for the destination coordinate system.

Returns:

Handle for use with **GDALReprojectionTransform()** (p. **??**), or NULL if the system fails to initialize the reprojection.

References GDALCreateReprojectionTransformer(), GDALDestroyReprojectionTransformer(), and GDALReprojectionTransform().

Referenced by GDALCreateGenImgProjTransformer2(), and GDALCreateReprojectionTransformer().

44.12.4.10 void* GDALCreateRPCTransformer (GDALRPCInfo * *psRPCInfo*, int *bReversed*, double *dfPixErrThreshold*, char ** *papszOptions*)

Create an RPC based transformer.

The geometric sensor model describing the physical relationship between image coordinates and ground coordinate is known as a Rigorous Projection Model. A Rigorous Projection Model expresses the mapping of the image space coordinates of rows and columns (r,c) onto the object space reference surface geodetic coordinates (long, lat, height).

RPC supports a generic description of the Rigorous Projection Models. The approximation used by GDAL (RPC00) is a set of rational polynomials exp ressing the normalized row and column values, (rn , cn), as a function of normalized geodetic latitude, longitude, and height, (P, L, H), given a set of normalized polynomial coefficients (LINE_NUM_COEF_n, LINE_DEN_COEF_n, SAMP_NUM_COEF_n, SAMP_-DEN_COEF_n). Normalized values, rather than actual values are used in order to minimize introduction of errors during the calculations. The transformation between row and column values (r,c), and normalized row and column values (rn, cn), and between the geodetic latitude, longitude, and height and normalized geodetic latitude, longitude, and height (P, L, H), is defined by a set of normalizing translations (offsets) and scales that ensure all values are contained i the range -1 to +1.

This function creates a GDALTransformFunc compatible transformer for going between image pixel/line and long/lat/height coordinates using RPCs. The RPCs are provided in a GDALRPCInfo structure which is normally read from metadata using GDALExtractRPCInfo().

GDAL RPC Metadata has the following entries (also described in GDAL RFC 22 and the GeoTIFF RPC document `http://geotiff.maptools.org/rpc_prop.html`.

- 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)

The transformer normally maps from pixel/line/height to long/lat/height space as a forward transformation though in RPC terms that would be considered an inverse transformation (and is solved by iterative approximation using long/lat/height to pixel/line transformations). The default direction can be reversed by passing bReversed=TRUE.

The iterative solution of pixel/line to lat/long/height is currently run for up to 10 iterations or until the apparent error is less than dfPixErrThreshold pixels. Passing zero will not avoid all error, but will cause the operation to run for the maximum number of iterations.

Additional options to the transformer can be supplied in papszOptions. Currently only one option is supported, though in the future more may be added, notably an option to extract elevation offsets from a DEM file.

Options:

- RPC_HEIGHT: a fixed height offset to be applied to all points passed in. In this situation the Z passed into the transformation function is assumed to be height above ground, and the RPC_HEIGHT is assumed to be an average height above sea level for ground in the target scene.

Parameters:

psRPCInfo Definition of the RPC parameters.

bReversed If true "forward" transformation will be lat/long to pixel/line instead of the normal pixel/line to lat/long.

dfPixErrThreshold the error (measured in pixels) allowed in the iterative solution of pixel/line to lat/long computations (the other way is always exact given the equations).

papszOptions Other transformer options (ie. RPC_HEIGHT=<z>).

Returns:

transformer callback data (deallocate with GDALDestroyTransformer()).

References CPLAtof(), GDALCreateRPCTransformer(), and GDALInvGeoTransform().

Referenced by GDALCreateGenImgProjTransformer2(), and GDALCreateRPCTransformer().

44.12.4.11 void* GDALCreateTPSTransformer (int *nGCPCount*, const GDAL_GCP * *pasGCPList*, int *bReversed*)

Create Thin Plate Spline transformer from GCPs.

The thin plate spline transformer produces exact transformation at all control points and smoothly varying transformations between control points with greatest influence from local control points. It is suitable for for many applications not well modelled by polynomial transformations.

Creating the TPS transformer involves solving systems of linear equations related to the number of control points involved. This solution is computed within this function call. It can be quite an expensive operation for large numbers of GCPs. For instance, for reference, it takes on the order of 10s for 400 GCPs on a 2GHz Athlon processor.

TPS Transformers are serializable.

The GDAL Thin Plate Spline transformer is based on code provided by Gilad Ronnen on behalf of VIZRT Inc (`http://www.visrt.com`). Incorporation of the algorithm into GDAL was supported by the Centro di Ecologia Alpina (`http://www.cealp.it`).

Parameters:

nGCPCount the number of GCPs in pasGCPList.

pasGCPList an array of GCPs to be used as input. bReversed

Returns:

the transform argument or NULL if creation fails.

References GDAL_GCP::dfGCPLine, GDAL_GCP::dfGCPPixel, GDAL_GCP::dfGCPX, GDAL_GCP::dfGCPY, GDALCreateTPSTransformer(), GDALDestroyTPSTransformer(), and GDALTPSTransform().

Referenced by GDALCreateGenImgProjTransformer2(), and GDALCreateTPSTransformer().

44.12.4.12 void GDALDestroyApproxTransformer (void * *pCBData*)

Cleanup approximate transformer.

Deallocates the resources allocated by **GDALCreateApproxTransformer()** (p. **??**).

Parameters:

pCBData callback data originally returned by **GDALCreateApproxTransformer()** (p. **??**).

References GDALDestroyApproxTransformer().

Referenced by GDALCreateApproxTransformer(), GDALDestroyApproxTransformer(), and GDALReprojectImage().

44.12.4.13 void GDALDestroyGCPTransformer (void * *pTransformArg*)

Destroy GCP transformer.

This function is used to destroy information about a GCP based polynomial transformation created with **GDALCreateGCPTransformer()** (p. **??**).

Parameters:

pTransformArg the transform arg previously returned by **GDALCreateGCPTransformer()** (p. **??**).

References GDALDestroyGCPTransformer().

Referenced by GDALCreateGCPTransformer(), GDALDestroyGCPTransformer(), and GDALDestroyGenImgProjTransformer().

44.12.4.14 void GDALDestroyGenImgProjTransformer (void * *hTransformArg*)

GenImgProjTransformer deallocator.

This function is used to deallocate the handle created with **GDALCreateGenImgProjTransformer()** (p. **??**).

Parameters:

hTransformArg the handle to deallocate.

References GDALDestroyGCPTransformer(), GDALDestroyGenImgProjTransformer(), GDALDestroyReprojectionTransformer(), and GDALDestroyTPSTransformer().

Referenced by GDALCreateGenImgProjTransformer2(), GDALDestroyGenImgProjTransformer(), and GDALReprojectImage().

44.12.4.15 void GDALDestroyReprojectionTransformer (void * *pTransformAlg*)

Destroy reprojection transformation.

Parameters:

pTransformArg the transformation handle returned by **GDALCreateReprojectionTransformer()** (p. **??**).

References GDALDestroyReprojectionTransformer().

Referenced by GDALCreateReprojectionTransformer(), GDALDestroyGenImgProjTransformer(), and GDALDestroyReprojectionTransformer().

44.12.4.16 void GDALDestroyTPSTransformer (void * *pTransformArg*)

Destroy TPS transformer.

This function is used to destroy information about a GCP based polynomial transformation created with **GDALCreateTPSTransformer()** (p. **??**).

Parameters:

pTransformArg the transform arg previously returned by **GDALCreateTPSTransformer()** (p. **??**).

References GDALDestroyTPSTransformer().

Referenced by GDALCreateTPSTransformer(), GDALDestroyGenImgProjTransformer(), and GDALDestroyTPSTransformer().

44.12.4.17 int GDALDitherRGB2PCT (GDALRasterBandH *hRed*, GDALRasterBandH *hGreen*, GDALRasterBandH *hBlue*, GDALRasterBandH *hTarget*, GDALColorTableH *hColorTable*, GDALProgressFunc *pfnProgress*, void * *pProgressArg*)

24bit to 8bit conversion with dithering.

This functions utilizes Floyd-Steinberg dithering in the process of converting a 24bit RGB image into a pseudocolored 8bit image using a provided color table.

The red, green and blue input bands do not necessarily need to come from the same file, but they must be the same width and height. They will be clipped to 8bit during reading, so non-eight bit bands are generally inappropriate. Likewise the hTarget band will be written with 8bit values and must match the width and height of the source bands.

The color table cannot have more than 256 entries.

Parameters:

hRed Red input band.

hGreen Green input band.

hBlue Blue input band.

hTarget Output band.

hColorTable the color table to use with the output band.

pfnProgress callback for reporting algorithm progress matching the GDALProgressFunc() semantics. May be NULL.

pProgressArg callback argument passed to pfnProgress.

Returns:

CE_None on success or CE_Failure if an error occurs.

References GDALColorEntry::c1, GDALColorEntry::c2, GDALColorEntry::c3, GDALDitherRGB2PCT(), GDALGetRasterBandXSize(), GDALGetRasterBandYSize(), GDALRasterIO(), GDT_Byte, GF_Read, and GF_Write.

Referenced by GDALDitherRGB2PCT().

44.12.4.18 int GDALGCPTransform (void * *pTransformArg*, int *bDstToSrc*, int *nPointCount*, double * *x*, double * *y*, double * *z*, int * *panSuccess*)

Transforms point based on GCP derived polynomial model.

This function matches the GDALTransformerFunc signature, and can be used to transform one or more points from pixel/line coordinates to georeferenced coordinates (SrcToDst) or vice versa (DstToSrc).

Parameters:

pTransformArg return value from **GDALCreateGCPTransformer()** (p. **??**).

bDstToSrc TRUE if transformation is from the destination (georeferenced) coordinates to pixel/line or FALSE when transforming from pixel/line to georeferenced coordinates.

nPointCount the number of values in the x, y and z arrays.

x array containing the X values to be transformed.

y array containing the Y values to be transformed.

z array containing the Z values to be transformed.

panSuccess array in which a flag indicating success (TRUE) or failure (FALSE) of the transformation are placed.

Returns:

TRUE.

References GDALGCPTransform().

Referenced by GDALCreateGCPTransformer(), GDALGCPTransform(), and GDALGenImgProjTransform().

44.12.4.19 int GDALGenImgProjTransform (void * *pTransformArg*, int *bDstToSrc*, int *nPointCount*, double * *padfX*, double * *padfY*, double * *padfZ*, int * *panSuccess*)

Perform general image reprojection transformation.

Actually performs the transformation setup in **GDALCreateGenImgProjTransformer()** (p. **??**). This function matches the signature required by the **GDALTransformerFunc()** (p. **??**), and more details on the arguments can be found in that topic.

References GDALGCPTransform(), GDALGenImgProjTransform(), GDALReprojectionTransform(), and GDALTPSTransform().

Referenced by GDALAutoCreateWarpedVRT(), GDALCreateGenImgProjTransformer2(), GDALGenImgProjTransform(), GDALRasterizeGeometries(), and GDALReprojectImage().

44.12.4.20 CPLErr GDALGridCreate (GDALGridAlgorithm *eAlgorithm*, const void * *poOptions*, GUInt32 *nPoints*, const double * *padfX*, const double * *padfY*, const double * *padfZ*, double *dfXMin*, double *dfXMax*, double *dfYMin*, double *dfYMax*, GUInt32 *nXSize*, GUInt32 *nYSize*, GDALDataType *eType*, void * *pData*, GDALProgressFunc *pfnProgress*, void * *pProgressArg*)

Create regular grid from the scattered data.

This fucntion takes the arrays of X and Y coordinates and corresponding Z values as input and computes regular grid (or call it a raster) from these scattered data. You should supply geometry and extent of the output grid and allocate array sufficient to hold such a grid.

Parameters:

eAlgorithm Gridding method.

poOptions Options to control choosen gridding method.

nPoints Number of elements in input arrays.

padfX Input array of X coordinates.

padfY Input array of Y coordinates.

padfZ Input array of Z values.

dfXMin Lowest X border of output grid.

dfXMax Highest X border of output grid.

dfYMin Lowest Y border of output grid.

dfYMax Highest Y border of output grid.

nXSize Number of columns in output grid.

nYSize Number of rows in output grid.

eType Data type of output array.

pData Pointer to array where the computed grid will be stored.

pfnProgress a GDALProgressFunc() compatible callback function for reporting progress or NULL.

pProgressArg argument to be passed to pfnProgress. May be NULL.

Returns:

CE_None on success or CE_Failure if something goes wrong.

References GDALGridCreate(), GDT_Byte, GDT_Float32, GDT_Float64, GDT_Int16, GDT_Int32, GDT_UInt16, GDT_UInt32, GGA_InverseDistanceToAPower, GGA_MetricMaximum, GGA_MetricMinimum, GGA_MetricRange, GGA_MovingAverage, and GGA_NearestNeighbor.

Referenced by GDALGridCreate().

44.12.4.21 CPLErr GDALPolygonize (GDALRasterBandH *hSrcBand*, GDALRasterBandH *hMaskBand*, OGRLayerH *hOutLayer*, int *iPixValField*, char ** *papszOptions*, GDALProgressFunc *pfnProgress*, void * *pProgressArg*)

Create polygon coverage from raster data.

This function creates vector polygons for all connected regions of pixels in the raster sharing a common pixel value. Optionally each polygon may be labelled with the pixel value in an attribute. Optionally a mask band can be provided to determine which pixels are eligible for processing.

Note that currently the source pixel band values are read into a signed 32bit integer buffer (Int32), so floating point or complex bands will be implicitly truncated before processing.

Polygon features will be created on the output layer, with polygon geometries representing the polygons. The polygon geometries will be in the georeferenced coordinate system of the image (based on the geotransform of the source dataset). It is acceptable for the output layer to already have features. Note that **GDALPolygonize()** (p. **??**) does not set the coordinate system on the output layer. Application code should do this when the layer is created, presumably matching the raster coordinate system.

The algorithm used attempts to minimize memory use so that very large rasters can be processed. However, if the raster has many polygons or very large/complex polygons, the memory use for holding polygon enumerations and active polygon geometries may grow to be quite large.

The algorithm will generally produce very dense polygon geometries, with edges that follow exactly on pixel boundaries for all non-interior pixels. For non-thematic raster data (such as satellite images) the result will essentially be one small polygon per pixel, and memory and output layer sizes will be substantial. The algorithm is primarily intended for relatively simple thematic imagery, masks, and classification results.

Parameters:

hSrcBand the source raster band to be processed.

hMaskBand an optional mask band. All pixels in the mask band with a value other than zero will be considered suitable for collection as polygons.

hOutLayer the vector feature layer to which the polygons should be written.

iPixValField the attribute field index indicating the feature attribute into which the pixel value of the polygon should be written.

papszOptions a name/value list of additional options (none currently supported).

pfnProgress callback for reporting algorithm progress matching the GDALProgressFunc() semantics. May be NULL.

pProgressArg callback argument passed to pfnProgress.

Returns:

CE_None on success or CE_Failure on a failure.

References GDALGetBandDataset(), GDALGetGeoTransform(), GDALGetRasterBandXSize(), GDALGetRasterBandYSize(), GDALPolygonize(), GDALRasterIO(), GDT_Int32, and GF_Read.

Referenced by GDALPolygonize().

44.12.4.22 CPLErr GDALRasterizeGeometries (GDALDatasetH *hDS*, int *nBandCount*, int * *panBandList*, int *nGeomCount*, OGRGeometryH * *pahGeometries*, GDALTransformerFunc *pfnTransformer*, void * *pTransformArg*, double * *padfGeomBurnValue*, char ** *papszOptions*, GDALProgressFunc *pfnProgress*, void * *pProgressArg*)

Burn geometries into raster.

Rasterize a list of geometric objects into a raster dataset. The geometries are passed as an array of OGRGeometryH handlers.

If the geometries are in the georferenced coordinates of the raster dataset, then the pfnTransform may be passed in NULL and one will be derived internally from the geotransform of the dataset. The transform needs to transform the geometry locations into pixel/line coordinates on the raster dataset.

The output raster may be of any GDAL supported datatype, though currently internally the burning is done either as GDT_Byte or GDT_Float32. This may be improved in the future. An explicit list of burn values for each geometry for each band must be passed in.

Parameters:

hDS output data, must be opened in update mode.

nBandCount the number of bands to be updated.

panBandList the list of bands to be updated.

nGeomCount the number of geometries being passed in pahGeometries.

pahGeometries the array of geometries to burn in.

pfnTransformer transformation to apply to geometries to put into pixel/line coordinates on raster. If NULL a geotransform based one will be created internally.

pTransformerArg callback data for transformer.

padfGeomBurnValue the array of values to burn into the raster. There should be nBandCount values for each geometry.

papszOption special options controlling rasterization, currently none are defined.

pfnProgress the progress function to report completion.

pProgressArg callback data for progress function.

Returns:

CE_None on success or CE_Failure on error.

References GDALCreateGenImgProjTransformer(), GDALGenImgProjTransform(), GDALGetDataTypeSize(), GDALRasterizeGeometries(), GDT_Byte, GDT_Float32, GDALDataset::GetRasterBand(), GDALRasterBand::GetRasterDataType(), GDALDataset::GetRasterXSize(), GDALDataset::GetRasterYSize(), GF_Read, GF_Write, and GDALDataset::RasterIO().

Referenced by GDALRasterizeGeometries().

44.12.4.23 int GDALReprojectionTransform (void * *pTransformArg*, int *bDstToSrc*, int *nPointCount*, double * *padfX*, double * *padfY*, double * *padfZ*, int * *panSuccess*)

Perform reprojection transformation.

Actually performs the reprojection transformation described in **GDALCreateReprojectionTransformer()** (p. **??**). This function matches the **GDALTransformerFunc()** (p. **??**) signature. Details of the arguments are described there.

References GDALReprojectionTransform().

Referenced by GDALCreateReprojectionTransformer(), GDALGenImgProjTransform(), and GDALReprojectionTransform().

44.12.4.24 void GDALSetGenImgProjTransformerDstGeoTransform (void * *hTransformArg*, const double * *padfGeoTransform*)

Set GenImgProj output geotransform.

Normally the "destination geotransform", or transformation between georeferenced output coordinates and pixel/line coordinates on the destination file is extracted from the destination file by **GDALCreateGenImgProjTransformer()** (p. **??**) and stored in the GenImgProj private info. However, sometimes it is inconvenient to have an output file handle with appropriate geotransform information when creating the transformation. For these cases, this function can be used to apply the destination geotransform.

Parameters:

hTransformArg the handle to update.

padfGeoTransform the destination geotransform to apply (six doubles).

References GDALInvGeoTransform(), and GDALSetGenImgProjTransformerDstGeoTransform().

Referenced by GDALAutoCreateWarpedVRT(), and GDALSetGenImgProjTransformerDstGeoTransform().

44.12.4.25 CPLErr GDALSieveFilter (GDALRasterBandH *hSrcBand*, GDALRasterBandH *hMaskBand*, GDALRasterBandH *hDstBand*, int *nSizeThreshold*, int *nConnectedness*, char ** *papszOptions*, GDALProgressFunc *pfnProgress*, void * *pProgressArg*)

Removes small raster polygons.

The function removes raster polygons smaller than a provided threshold size (in pixels) and replaces replaces them with the pixel value of the largest neighbour polygon.

Polygon are determined (per GDALRasterPolygonEnumerator) as regions of the raster where the pixels all have the same value, and that are contiguous (connected).

Pixels determined to be "nodata" per hMaskBand will not be treated as part of a polygon regardless of their pixel values. Nodata areas will never be changed nor affect polygon sizes.

Polygons smaller than the threshold with no neighbours that are as large as the threshold will not be altered. Polygons surrounded by nodata areas will therefore not be altered.

The algorithm makes three passes over the input file to enumerate the polygons and collect limited information about them. Memory use is proportional to the number of polygons (roughly 24 bytes per polygon), but is not directly related to the size of the raster. So very large raster files can be processed effectively if there aren't too many polygons. But extremely noisy rasters with many one pixel polygons will end up being expensive (in memory) to process.

Parameters:

hSrcBand the source raster band to be processed.

hMaskBand an optional mask band. All pixels in the mask band with a value other than zero will be considered suitable for inclusion in polygons.

hDstBand the output raster band. It may be the same as hSrcBand to update the source in place.

nSizeThreshold raster polygons with sizes smaller than this will be merged into their largest neighbour.

nConnectedness either 4 indicating that diagonal pixels are not considered directly adjacent for polygon membership purposes or 8 indicating they are.

papszOption algorithm options in name=value list form. None currently supported.

pfnProgress callback for reporting algorithm progress matching the GDALProgressFunc() semantics. May be NULL.

pProgressArg callback argument passed to pfnProgress.

Returns:

CE_None on success or CE_Failure if an error occurs.

References GDALGetRasterBandXSize(), GDALGetRasterBandYSize(), GDALRasterIO(), GDALSieveFilter(), GDT_Int32, GF_Read, and GF_Write.

Referenced by GDALSieveFilter().

44.12.4.26 int GDALSimpleImageWarp (GDALDatasetH *hSrcDS*, GDALDatasetH *hDstDS*, int *nBandCount*, int * *panBandList*, GDALTransformerFunc *pfnTransform*, void * *pTransformArg*, GDALProgressFunc *pfnProgress*, void * *pProgressArg*, char ** *papszWarpOptions*)

Perform simple image warp.

Copies an image from a source dataset to a destination dataset applying an application defined transformation. This algorithm is called simple because it lacks many options such as resampling kernels (other than nearest neighbour), support for data types other than 8bit, and the ability to warp images without holding the entire source and destination image in memory.

The following option(s) may be passed in papszWarpOptions.

- "INIT=v[,v...]": This option indicates that the output dataset should be initialized to the indicated value in any area valid data is not written. Distinct values may be listed for each band separated by columns.

Parameters:

hSrcDS the source image dataset.

hDstDS the destination image dataset.

nBandCount the number of bands to be warped. If zero, all bands will be processed.

panBandList the list of bands to translate.

pfnTransform the transformation function to call. See **GDALTransformerFunc()** (p. **??**).

pTransformArg the callback handle to pass to pfnTransform.

pfnProgress the function used to report progress. See GDALProgressFunc().

pProgressArg the callback handle to pass to pfnProgress.

papszWarpOptions additional options controlling the warp.

Returns:

TRUE if the operation completes, or FALSE if an error occurs.

References GDALGetRasterBand(), GDALGetRasterCount(), GDALGetRasterXSize(), GDALGetRasterYSize(), GDALRasterIO(), GDALSimpleImageWarp(), GDT_Byte, GF_Read, and GF_Write.

Referenced by GDALSimpleImageWarp().

44.12.4.27 CPLErr GDALSuggestedWarpOutput (GDALDatasetH *hSrcDS*, GDALTransformerFunc *pfnTransformer*, void * *pTransformArg*, double * *padfGeoTransformOut*, int * *pnPixels*, int * *pnLines*)

Suggest output file size.

This function is used to suggest the size, and georeferenced extents appropriate given the indicated transformation and input file. It walks the edges of the input file (approximately 20 sample points along each edge) transforming into output coordinates in order to get an extents box.

Then a resolution is computed with the intent that the length of the distance from the top left corner of the output imagery to the bottom right corner would represent the same number of pixels as in the source image. Note that if the image is somewhat rotated the diagonal taken isnt of the whole output bounding rectangle, but instead of the locations where the top/left and bottom/right corners transform. The output pixel size is always square. This is intended to approximately preserve the resolution of the input data in the output file.

The values returned in padfGeoTransformOut, pnPixels and pnLines are the suggested number of pixels and lines for the output file, and the geotransform relating those pixels to the output georeferenced coordinates.

The trickiest part of using the function is ensuring that the transformer created is from source file pixel/line coordinates to output file georeferenced coordinates. This can be accomplished with GDALCreateGenImProjTransformer() by passing a NULL for the hDstDS.

Parameters:

hSrcDS the input image (it is assumed the whole input images is being transformed).

pfnTransformer the transformer function.

pTransformArg the callback data for the transformer function.

padfGeoTransformOut the array of six doubles in which the suggested geotransform is returned.

pnPixels int in which the suggest pixel width of output is returned.

pnLines int in which the suggest pixel height of output is returned.

Returns:

CE_None if successful or CE_Failure otherwise.

References GDALSuggestedWarpOutput(), and GDALSuggestedWarpOutput2().

Referenced by GDALAutoCreateWarpedVRT(), and GDALSuggestedWarpOutput().

44.12.4.28 CPLErr GDALSuggestedWarpOutput2 (GDALDatasetH *hSrcDS*, GDALTransformerFunc *pfnTransformer*, void * *pTransformArg*, double * *padfGeoTransformOut*, int * *pnPixels*, int * *pnLines*, double * *padfExtent*, int *nOptions*)

Suggest output file size.

This function is used to suggest the size, and georeferenced extents appropriate given the indicated transformation and input file. It walks the edges of the input file (approximately 20 sample points along each edge) transforming into output coordinates in order to get an extents box.

Then a resolution is computed with the intent that the length of the distance from the top left corner of the output imagery to the bottom right corner would represent the same number of pixels as in the source image. Note that if the image is somewhat rotated the diagonal taken isnt of the whole output bounding rectangle, but instead of the locations where the top/left and bottom/right corners transform. The output

pixel size is always square. This is intended to approximately preserve the resolution of the input data in the output file.

The values returned in padfGeoTransformOut, pnPixels and pnLines are the suggested number of pixels and lines for the output file, and the geotransform relating those pixels to the output georeferenced coordinates.

The trickiest part of using the function is ensuring that the transformer created is from source file pixel/line coordinates to output file georeferenced coordinates. This can be accomplished with GDALCreateGenImProjTransformer() by passing a NULL for the hDstDS.

Parameters:

hSrcDS the input image (it is assumed the whole input images is being transformed).

pfnTransformer the transformer function.

pTransformArg the callback data for the transformer function.

padfGeoTransformOut the array of six doubles in which the suggested geotransform is returned.

pnPixels int in which the suggest pixel width of output is returned.

pnLines int in which the suggest pixel height of output is returned.

padfExtent Four entry array to return extents as (xmin, ymin, xmax, ymax).

nOptions Options, currently always zero.

Returns:

CE_None if successful or CE_Failure otherwise.

References GDALGetRasterXSize(), GDALGetRasterYSize(), and GDALSuggestedWarpOutput2().

Referenced by GDALSuggestedWarpOutput(), and GDALSuggestedWarpOutput2().

44.12.4.29 int GDALTPSTransform (void * *pTransformArg*, int *bDstToSrc*, int *nPointCount*, double * *x*, double * *y*, double * *z*, int * *panSuccess*)

Transforms point based on GCP derived polynomial model.

This function matches the GDALTransformerFunc signature, and can be used to transform one or more points from pixel/line coordinates to georeferenced coordinates (SrcToDst) or vice versa (DstToSrc).

Parameters:

pTransformArg return value from **GDALCreateTPSTransformer()** (p. **??**).

bDstToSrc TRUE if transformation is from the destination (georeferenced) coordinates to pixel/line or FALSE when transforming from pixel/line to georeferenced coordinates.

nPointCount the number of values in the x, y and z arrays.

x array containing the X values to be transformed.

y array containing the Y values to be transformed.

z array containing the Z values to be transformed.

panSuccess array in which a flag indicating success (TRUE) or failure (FALSE) of the transformation are placed.

Returns:

TRUE.

References GDALTPSTransform().

Referenced by GDALCreateTPSTransformer(), GDALGenImgProjTransform(), and GDALTPSTransform().

44.13 gdal_vrt.h File Reference

```
#include "gdal.h"
#include "cpl_port.h"
#include "cpl_error.h"
#include "cpl_minixml.h"
```

Defines

- #define **VRT_NODATA_UNSET** -1234.56

Typedefs

- typedef CPLErr(∗ **VRTImageReadFunc**)(void ∗hCBData, int nXOff, int nYOff, int nXSize, int nYSize, void ∗pData)
- typedef void ∗ **VRTDriverH**
- typedef void ∗ **VRTSourceH**
- typedef void ∗ **VRTSimpleSourceH**
- typedef void ∗ **VRTAveragedSourceH**
- typedef void ∗ **VRTComplexSourceH**
- typedef void ∗ **VRTFilteredSourceH**
- typedef void ∗ **VRTKernelFilteredSourceH**
- typedef void ∗ **VRTAverageFilteredSourceH**
- typedef void ∗ **VRTFuncSourceH**
- typedef void ∗ **VRTDatasetH**
- typedef void ∗ **VRTWarpedDatasetH**
- typedef void ∗ **VRTRasterBandH**
- typedef void ∗ **VRTSourcedRasterBandH**
- typedef void ∗ **VRTWarpedRasterBandH**
- typedef void ∗ **VRTDerivedRasterBandH**
- typedef void ∗ **VRTRawRasterBandH**

Functions

- void **GDALRegister_VRT** (void)
- VRTDatasetH **VRTCreate** (int, int)
- void **VRTFlushCache** (VRTDatasetH)
- **CPLXMLNode** ∗ **VRTSerializeToXML** (VRTDatasetH, const char ∗)
- int **VRTAddBand** (VRTDatasetH, **GDALDataType**, char ∗∗)
- CPLErr **VRTAddSource** (VRTSourcedRasterBandH, VRTSourceH)
- CPLErr **VRTAddSimpleSource** (VRTSourcedRasterBandH, GDALRasterBandH, int, int, int, int, int, int, int, int, const char ∗, double)
- CPLErr **VRTAddComplexSource** (VRTSourcedRasterBandH, GDALRasterBandH, int, int, int, int, int, int, int, int, double, double, double)
- CPLErr **VRTAddFuncSource** (VRTSourcedRasterBandH, VRTImageReadFunc, void ∗, double)

44.13.1 Detailed Description

Public (C callable) entry points for virtual GDAL dataset objects.

44.13.2 Function Documentation

44.13.2.1 int VRTAddBand (VRTDatasetH *hDataset*, GDALDataType *eType*, char ** *papszOptions*)

See also:

VRTDataset::VRTAddBand() (p. **??**).

References VRTAddBand().

Referenced by VRTAddBand().

44.13.2.2 CPLErr VRTAddComplexSource (VRTSourcedRasterBandH *hVRTBand*, GDALRasterBandH *hSrcBand*, int *nSrcXOff*, int *nSrcYOff*, int *nSrcXSize*, int *nSrcYSize*, int *nDstXOff*, int *nDstYOff*, int *nDstXSize*, int *nDstYSize*, double *dfScaleOff*, double *dfScaleRatio*, double *dfNoDataValue*)

See also:

VRTSourcedRasterBand::AddComplexSource().

References VRTAddComplexSource().

Referenced by VRTAddComplexSource().

44.13.2.3 CPLErr VRTAddFuncSource (VRTSourcedRasterBandH *hVRTBand*, VRTImageReadFunc *pfnReadFunc*, void * *pCBData*, double *dfNoDataValue*)

See also:

VRTSourcedRasterBand::AddFuncSource().

References VRTAddFuncSource().

Referenced by VRTAddFuncSource().

44.13.2.4 CPLErr VRTAddSimpleSource (VRTSourcedRasterBandH *hVRTBand*, GDALRasterBandH *hSrcBand*, int *nSrcXOff*, int *nSrcYOff*, int *nSrcXSize*, int *nSrcYSize*, int *nDstXOff*, int *nDstYOff*, int *nDstXSize*, int *nDstYSize*, const char * *pszResampling*, double *dfNoDataValue*)

See also:

VRTSourcedRasterBand::AddSimpleSource().

References VRTAddSimpleSource().

Referenced by VRTAddSimpleSource().

44.13.2.5 CPLErr VRTAddSource (VRTSourcedRasterBandH *hVRTBand*, VRTSourceH *hNewSource*)

See also:

VRTSourcedRasterBand::AddSource().

References VRTAddSource().

Referenced by VRTAddSource().

44.13.2.6 VRTDatasetH VRTCreate (int *nXSize*, int *nYSize*)

See also:

VRTDataset::VRTDataset()

References VRTCreate().

Referenced by VRTCreate().

44.13.2.7 void VRTFlushCache (VRTDatasetH *hDataset*)

See also:

VRTDataset::FlushCache()

References VRTFlushCache().

Referenced by VRTFlushCache().

44.13.2.8 CPLXMLNode* VRTSerializeToXML (VRTDatasetH *hDataset*, const char * *pszVRTPath*)

See also:

VRTDataset::SerializeToXML()

References VRTSerializeToXML().

Referenced by VRTSerializeToXML().

44.14 gdalgrid.h File Reference

```
#include "gdal_alg.h"
```

Typedefs

- typedef CPLErr(∗ **GDALGridFunction**)(const void ∗, GUInt32, const double ∗, const double ∗, const double ∗, double, double, double ∗)

Functions

- CPLErr **GDALGridInverseDistanceToAPower** (const void ∗, GUInt32, const double ∗, const double ∗, const double ∗, double, double, double ∗)
- CPLErr **GDALGridInverseDistanceToAPowerNoSearch** (const void ∗, GUInt32, const double ∗, const double ∗, const double ∗, double, double, double ∗)
- CPLErr **GDALGridMovingAverage** (const void ∗, GUInt32, const double ∗, const double ∗, const double ∗, double, double, double ∗)
- CPLErr **GDALGridNearestNeighbor** (const void ∗, GUInt32, const double ∗, const double ∗, const double ∗, double, double, double ∗)
- CPLErr **GDALGridDataMetricMinimum** (const void ∗, GUInt32, const double ∗, const double ∗, const double ∗, double, double, double ∗)
- CPLErr **GDALGridDataMetricMaximum** (const void ∗, GUInt32, const double ∗, const double ∗, const double ∗, double, double, double ∗)
- CPLErr **GDALGridDataMetricRange** (const void ∗, GUInt32, const double ∗, const double ∗, const double ∗, double, double, double ∗)

44.14.1 Detailed Description

GDAL gridder related entry points and definitions.

44.14.2 Function Documentation

44.14.2.1 CPLErr GDALGridDataMetricMaximum (const void ∗ *poOptions*, GUInt32 *nPoints*, const double ∗ *padfX*, const double ∗ *padfY*, const double ∗ *padfZ*, double *dfXPoint*, double *dfYPoint*, double ∗ *pdfValue*)

Maximum data value (data metric).

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.

Parameters:

poOptions Algorithm parameters. This should point to **GDALGridDataMetricsOptions** (p. **??**) object.

nPoints Number of elements in input arrays.

padfX Input array of X coordinates.

padfY Input array of Y coordinates.

padfZ Input array of Z values.

dfXPoint X coordinate of the point to compute.

dfYPoint Y coordinate of the point to compute.

pdfValue Pointer to variable where the computed grid node value will be returned.

Returns:

CE_None on success or CE_Failure if something goes wrong.

44.14.2.2 CPLErr GDALGridDataMetricMinimum (const void * *poOptions*, GUInt32 *nPoints*, const double * *padfX*, const double * *padfY*, const double * *padfZ*, double *dfXPoint*, double *dfYPoint*, double * *pdfValue*)

Minimum data value (data metric).

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

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

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.

Parameters:

poOptions Algorithm parameters. This should point to **GDALGridDataMetricsOptions** (p. **??**) object.

nPoints Number of elements in input arrays.

padfX Input array of X coordinates.

padfY Input array of Y coordinates.

padfZ Input array of Z values.

dfXPoint X coordinate of the point to compute.

dfYPoint Y coordinate of the point to compute.

pdfValue Pointer to variable where the computed grid node value will be returned.

Returns:

CE_None on success or CE_Failure if something goes wrong.

44.14.2.3 CPLErr GDALGridDataMetricRange (const void * *poOptions*, GUInt32 *nPoints*, const double * *padfX*, const double * *padfY*, const double * *padfZ*, double *dfXPoint*, double *dfYPoint*, double * *pdfValue*)

Data range (data metric).

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.

Parameters:

poOptions Algorithm parameters. This should point to **GDALGridDataMetricsOptions** (p. **??**) object.

nPoints Number of elements in input arrays.

padfX Input array of X coordinates.

padfY Input array of Y coordinates.

padfZ Input array of Z values.

dfXPoint X coordinate of the point to compute.

dfYPoint Y coordinate of the point to compute.

pdfValue Pointer to variable where the computed grid node value will be returned.

Returns:

CE_None on success or CE_Failure if something goes wrong.

44.14.2.4 CPLErr GDALGridInverseDistanceToAPower (const void * *poOptions*, GUInt32 *nPoints*, const double * *padfX*, const double * *padfY*, const double * *padfZ*, double *dfXPoint*, double *dfYPoint*, double * *pdfValue*)

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

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

Parameters:

poOptions Algorithm parameters. This should point to **GDALGridInverseDistanceToAPowerOptions** (p. **??**) object.

nPoints Number of elements in input arrays.

padfX Input array of X coordinates.

padfY Input array of Y coordinates.

padfZ Input array of Z values.

dfXPoint X coordinate of the point to compute.

dfYPoint Y coordinate of the point to compute.

nXPoint X position of the point to compute.

nYPoint Y position of the point to compute.

pdfValue Pointer to variable where the computed grid node value will be returned.

Returns:

CE_None on success or CE_Failure if something goes wrong.

44.14.2.5 CPLErr GDALGridInverseDistanceToAPowerNoSearch (const void * *poOptions*, GUInt32 *nPoints*, const double * *padfX*, const double * *padfY*, const double * *padfZ*, double *dfXPoint*, double *dfYPoint*, double * *pdfValue*)

Inverse distance to a power for whole data set.

This is somewhat optimized version of the Inverse Distance to a Power method. It is used when the search ellips is not set. The algorithm and parameters are the same as in **GDALGridInverseDistanceToAPower()** (p. **??**), but this implementation works faster, because of no search.

See also:

GDALGridInverseDistanceToAPower() (p. **??**)

44.14.2.6 CPLErr GDALGridMovingAverage (const void * *poOptions*, GUInt32 *nPoints*, const double * *padfX*, const double * *padfY*, const double * *padfZ*, double *dfXPoint*, double *dfYPoint*, double * *pdfValue*)

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 ellipse.

Parameters:

poOptions Algorithm parameters. This should point to **GDALGridMovingAverageOptions** (p. **??**) object.

nPoints Number of elements in input arrays.

padfX Input array of X coordinates.

padfY Input array of Y coordinates.

padfZ Input array of Z values.

dfXPoint X coordinate of the point to compute.

dfYPoint Y coordinate of the point to compute.

pdfValue Pointer to variable where the computed grid node value will be returned.

Returns:

CE_None on success or CE_Failure if something goes wrong.

44.14.2.7 CPLErr GDALGridNearestNeighbor (const void * *poOptions*, GUInt32 *nPoints*, const double * *padfX*, const double * *padfY*, const double * *padfZ*, double *dfXPoint*, double *dfYPoint*, double * *pdfValue*)

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.

Parameters:

poOptions Algorithm parameters. This should point to **GDALGridNearestNeighborOptions** (p. **??**) object.

nPoints Number of elements in input arrays.

padfX Input array of X coordinates.

padfY Input array of Y coordinates.

padfZ Input array of Z values.

dfXPoint X coordinate of the point to compute.

dfYPoint Y coordinate of the point to compute.

pdfValue Pointer to variable where the computed grid node value will be returned.

Returns:

CE_None on success or CE_Failure if something goes wrong.

44.15 gdalwarper.h File Reference

```
#include "gdal_alg.h"
#include "cpl_minixml.h"
```

Classes

- struct **GDALWarpOptions**
- class **GDALWarpKernel**
- class **GDALWarpOperation**

Typedefs

- typedef int(∗ **GDALMaskFunc**)(void ∗pMaskFuncArg, int nBandCount, **GDALDataType** eType, int nXOff, int nYOff, int nXSize, int nYSize, GByte ∗∗papabyImageData, int bMaskIsFloat, void ∗pMask)
- typedef void ∗ **GDALWarpOperationH**

Enumerations

- enum **GDALResampleAlg** {

  GRA_NearestNeighbour = 0, **GRA_Bilinear** = 1, **GRA_Cubic** = 2, **GRA_CubicSpline** = 3, **GRA_Lanczos** = 4 }

Functions

- CPLErr **GDALWarpNoDataMasker** (void ∗pMaskFuncArg, int nBandCount, **GDALDataType** eType, int nXOff, int nYOff, int nXSize, int nYSize, GByte ∗∗papabyImageData, int bMaskIsFloat, void ∗pValidityMask)
- CPLErr **GDALWarpDstAlphaMasker** (void ∗pMaskFuncArg, int nBandCount, **GDALDataType** eType, int nXOff, int nYOff, int nXSize, int nYSize, GByte ∗∗, int bMaskIsFloat, void ∗pValidityMask)
- CPLErr **GDALWarpSrcAlphaMasker** (void ∗pMaskFuncArg, int nBandCount, **GDALDataType** eType, int nXOff, int nYOff, int nXSize, int nYSize, GByte ∗∗, int bMaskIsFloat, void ∗pValidityMask)
- CPLErr **GDALWarpCutlineMasker** (void ∗pMaskFuncArg, int nBandCount, **GDALDataType** eType, int nXOff, int nYOff, int nXSize, int nYSize, GByte ∗∗, int bMaskIsFloat, void ∗pValidityMask)
- **GDALWarpOptions** ∗ **GDALCreateWarpOptions** (void)
- void **GDALDestroyWarpOptions** (**GDALWarpOptions** ∗)
- **GDALWarpOptions** ∗ **GDALCloneWarpOptions** (const **GDALWarpOptions** ∗)
- **CPLXMLNode** ∗ **GDALSerializeWarpOptions** (const **GDALWarpOptions** ∗)
- **GDALWarpOptions** ∗ **GDALDeserializeWarpOptions** (**CPLXMLNode** ∗)
- CPLErr **GDALReprojectImage** (GDALDatasetH hSrcDS, const char ∗pszSrcWKT, GDALDatasetH hDstDS, const char ∗pszDstWKT, **GDALResampleAlg** eResampleAlg, double dfWarpMemoryLimit, double dfMaxError, GDALProgressFunc pfnProgress, void ∗pProgressArg, **GDALWarpOptions** ∗psOptions)

- CPLErr **GDALCreateAndReprojectImage** (GDALDatasetH hSrcDS, const char ∗pszSrcWKT, const char ∗pszDstFilename, const char ∗pszDstWKT, GDALDriverH hDstDriver, char ∗∗papszCreateOptions, **GDALResampleAlg** eResampleAlg, double dfWarpMemoryLimit, double dfMaxError, GDALProgressFunc pfnProgress, void ∗pProgressArg, **GDALWarpOptions** ∗psOptions)
- GDALDatasetH **GDALAutoCreateWarpedVRT** (GDALDatasetH hSrcDS, const char ∗pszSrcWKT, const char ∗pszDstWKT, **GDALResampleAlg** eResampleAlg, double dfMaxError, const **GDALWarpOptions** ∗psOptions)
- GDALDatasetH **GDALCreateWarpedVRT** (GDALDatasetH hSrcDS, int nPixels, int nLines, double ∗padfGeoTransform, **GDALWarpOptions** ∗psOptions)
- CPLErr **GDALInitializeWarpedVRT** (GDALDatasetH hDS, **GDALWarpOptions** ∗psWO)
- GDALWarpOperationH **GDALCreateWarpOperation** (const **GDALWarpOptions** ∗)
- void **GDALDestroyWarpOperation** (GDALWarpOperationH)
- CPLErr **GDALChunkAndWarpImage** (GDALWarpOperationH, int, int, int, int)
- CPLErr **GDALChunkAndWarpMulti** (GDALWarpOperationH, int, int, int, int)
- CPLErr **GDALWarpRegion** (GDALWarpOperationH, int, int, int, int, int, int, int, int)
- CPLErr **GDALWarpRegionToBuffer** (GDALWarpOperationH, int, int, int, int, void ∗, **GDALDataType**, int, int, int, int)

44.15.1 Detailed Description

GDAL warper related entry points and definitions. Eventually it is expected that this file will be mostly private to the implementation, and the public C entry points will be available in **gdal_alg.h** (p. **??**).

44.15.2 Enumeration Type Documentation

44.15.2.1 enum GDALResampleAlg

Warp Resampling Algorithm

Enumerator:

GRA_NearestNeighbour Nearest neighbour (select on one input pixel)

GRA_Bilinear Bilinear (2x2 kernel)

GRA_Cubic Cubic Convolution Approximation (4x4 kernel)

GRA_CubicSpline Cubic B-Spline Approximation (4x4 kernel)

GRA_Lanczos Lanczos windowed sinc interpolation (6x6 kernel)

44.15.3 Function Documentation

44.15.3.1 GDALDatasetH GDALAutoCreateWarpedVRT (GDALDatasetH *hSrcDS*, const char ∗ *pszSrcWKT*, const char ∗ *pszDstWKT*, GDALResampleAlg *eResampleAlg*, double *dfMaxError*, const GDALWarpOptions ∗ *psOptionsIn*)

Create virtual warped dataset automatically.

This function will create a warped virtual file representing the input image warped into the target coordinate system. A GenImgProj transformation is created to accomplish any required GCP/Geotransform warp and reprojection to the target coordinate system. The output virtual dataset will be "northup" in the target

coordinate system. The **GDALSuggestedWarpOutput()** (p. **??**) function is used to determine the bounds and resolution of the output virtual file which should be large enough to include all the input image

Note that the constructed GDALDatasetH will acquire one or more references to the passed in hSrcDS. Reference counting semantics on the source dataset should be honoured. That is, don't just **GDALClose()** (p. **??**) it unless it was opened with **GDALOpenShared()** (p. **??**).

The returned dataset will have no associated filename for itself. If you want to write the virtual dataset description to a file, use the **GDALSetDescription()** (p. **??**) function (or SetDescription() method) on the dataset to assign a filename before it is closed.

Parameters:

hSrcDS The source dataset.

pszSrcWKT The coordinate system of the source image. If NULL, it will be read from the source image.

pszDstWKT The coordinate system to convert to. If NULL no change of coordinate system will take place.

eResampleAlg One of GRA_NearestNeighbour, GRA_Bilinear, GRA_Cubic or GRA_CubicSpline. Controls the sampling method used.

dfMaxError Maximum error measured in input pixels that is allowed in approximating the transformation (0.0 for exact calculations).

psOptions Additional warp options, normally NULL.

Returns:

NULL on failure, or a new virtual dataset handle on success.

References GDALWarpOptions::eResampleAlg, GDALApproxTransform(), GDALAutoCreateWarpedVRT(), GDALCreateApproxTransformer(), GDALCreateGenImgProjTransformer(), GDALCreateWarpedVRT(), GDALGenImgProjTransform(), GDALGetGCPCount(), GDALGetGCPProjection(), GDALGetProjectionRef(), GDALGetRasterCount(), GDALSetGenImgProjTransformerDstGeoTransform(), GDALSetProjection(), GDALSuggestedWarpOutput(), GDALWarpOptions::hSrcDS, GDALWarpOptions::nBandCount, GDALWarpOptions::panDstBands, GDALWarpOptions::panSrcBands, GDALWarpOptions::pfnTransformer, and GDALWarpOptions::pTransformerArg.

Referenced by GDALAutoCreateWarpedVRT().

44.15.3.2 CPLErr GDALChunkAndWarpImage (GDALWarpOperationH *hOperation*, int *nDstXOff*, int *nDstYOff*, int *nDstXSize*, int *nDstYSize*)

See also:

GDALWarpOperation::ChunkAndWarpImage() (p. **??**)

References GDALWarpOperation::ChunkAndWarpImage(), and GDALChunkAndWarpImage().

Referenced by GDALChunkAndWarpImage().

44.15.3.3 CPLErr GDALChunkAndWarpMulti (GDALWarpOperationH *hOperation*, int *nDstXOff*, int *nDstYOff*, int *nDstXSize*, int *nDstYSize*)

See also:

GDALWarpOperation::ChunkAndWarpMulti() (p. **??**)

References GDALWarpOperation::ChunkAndWarpMulti(), and GDALChunkAndWarpMulti().

Referenced by GDALChunkAndWarpMulti().

44.15.3.4 GDALDatasetH GDALCreateWarpedVRT (GDALDatasetH *hSrcDS*, int *nPixels*, int *nLines*, double * *padfGeoTransform*, GDALWarpOptions * *psOptions*)

Create virtual warped dataset.

This function will create a warped virtual file representing the input image warped based on a provided transformation. Output bounds and resolution are provided explicitly.

Note that the constructed GDALDatasetH will acquire one or more references to the passed in hSrcDS. Reference counting semantics on the source dataset should be honoured. That is, don't just **GDALClose()** (p. **??**) it unless it was opened with **GDALOpenShared()** (p. **??**).

Parameters:

hSrcDS The source dataset.

nOverviewLevels The number of "power of 2" overview levels to be built. If zero, no overview levels will be managed.

psOptions Additional warp options, normally NULL.

Returns:

NULL on failure, or a new virtual dataset handle on success.

References GDALCreateWarpedVRT(), GDALGetRasterBand(), GDALRasterBand::GetRasterDataType(), GDALWarpOptions::hDstDS, and GDALWarpOptions::nBandCount.

Referenced by GDALAutoCreateWarpedVRT(), and GDALCreateWarpedVRT().

44.15.3.5 GDALWarpOperationH GDALCreateWarpOperation (const GDALWarpOptions * *psNewOptions*)

See also:

GDALWarpOperation::Initialize() (p. **??**)

References GDALCreateWarpOperation(), and GDALWarpOperation::Initialize().

Referenced by GDALCreateWarpOperation().

44.15.3.6 void GDALDestroyWarpOperation (GDALWarpOperationH *hOperation*)

See also:

GDALWarpOperation::~GDALWarpOperation()

References GDALDestroyWarpOperation().

Referenced by GDALDestroyWarpOperation().

44.15.3.7 CPLErr GDALInitializeWarpedVRT (GDALDatasetH *hDS*, GDALWarpOptions * *psWO*)

Set warp info on virtual warped dataset.

Initializes all the warping information for a virtual warped dataset.

This method is the same as the C++ method VRTWarpedDataset::Initialize().

Parameters:

hDS dataset previously created with the VRT driver, and a SUBCLASS of "VRTWarpedDataset".

psWO the warp options to apply. Note that ownership of the transformation information is taken over by the function though everything else remains the property of the caller.

Returns:

CE_None on success or CE_Failure if an error occurs.

References GDALInitializeWarpedVRT().

Referenced by GDALInitializeWarpedVRT().

44.15.3.8 CPLErr GDALReprojectImage (GDALDatasetH *hSrcDS*, const char * *pszSrcWKT*, GDALDatasetH *hDstDS*, const char * *pszDstWKT*, GDALResampleAlg *eResampleAlg*, double *dfWarpMemoryLimit*, double *dfMaxError*, GDALProgressFunc *pfnProgress*, void * *pProgressArg*, GDALWarpOptions * *psOptions*)

Reproject image.

This is a convenience function utilizing the **GDALWarpOperation** (p. **??**) class to reproject an image from a source to a destination. In particular, this function takes care of establishing the transformation function to implement the reprojection, and will default a variety of other warp options.

By default all bands are transferred, with no masking or nodata values in effect. No metadata, projection info, or color tables are transferred to the output file.

Parameters:

hSrcDS the source image file.

pszSrcWKT the source projection. If NULL the source projection is read from from hSrcDS.

hDstDS the destination image file.

pszDstWKT the destination projection. If NULL the destination projection will be read from hDstDS.

eResampleAlg the type of resampling to use.

dfWarpMemoryLimit the amount of memory (in bytes) that the warp API is allowed to use for caching. This is in addition to the memory already allocated to the GDAL caching (as per **GDALSetCacheMax()** (p. **??**)). May be 0.0 to use default memory settings.

dfMaxError maximum error measured in input pixels that is allowed in approximating the transformation (0.0 for exact calculations).

pfnProgress a GDALProgressFunc() compatible callback function for reporting progress or NULL.

pProgressArg argument to be passed to pfnProgress. May be NULL.

psOptions warp options, normally NULL.

Returns:

CE_None on success or CE_Failure if something goes wrong.

References GDALWarpOperation::ChunkAndWarpImage(), GDALWarpOptions::eResampleAlg, GDALApproxTransform(), GDALCreateApproxTransformer(), GDALCreateGenImgProjTransformer(), GDALDestroyApproxTransformer(), GDALDestroyGenImgProjTransformer(), GDALGenImgProjTransform(), GDALGetRasterBand(), GDALGetRasterCount(), GDALGetRasterNoDataValue(), GDALGetRasterXSize(), GDALGetRasterYSize(), GDALWarpOptions::hDstDS, GDALWarpOptions::hSrcDS, GDALWarpOperation::Initialize(), GDALWarpOptions::nBandCount, GDALWarpOptions::padfSrcNoDataImag, GDALWarpOptions::padfSrcNoDataReal, GDALWarpOptions::panDstBands, GDALWarpOptions::panSrcBands, GDALWarpOptions::pfnProgress, GDALWarpOptions::pfnTransformer, GDALWarpOptions::pProgressArg, and GDALWarpOptions::pTransformerArg.

44.15.3.9 CPLErr GDALWarpRegion (GDALWarpOperationH *hOperation*, int *nDstXOff*, int *nDstYOff*, int *nDstXSize*, int *nDstYSize*, int *nSrcXOff*, int *nSrcYOff*, int *nSrcXSize*, int *nSrcYSize*)

See also:

GDALWarpOperation::WarpRegion() (p. **??**)

References GDALWarpRegion(), and GDALWarpOperation::WarpRegion().

Referenced by GDALWarpRegion().

44.15.3.10 CPLErr GDALWarpRegionToBuffer (GDALWarpOperationH *hOperation*, int *nDstXOff*, int *nDstYOff*, int *nDstXSize*, int *nDstYSize*, void * *pDataBuf*, GDALDataType *eBufDataType*, int *nSrcXOff*, int *nSrcYOff*, int *nSrcXSize*, int *nSrcYSize*)

See also:

GDALWarpOperation::WarpRegionToBuffer() (p. **??**)

References GDALWarpRegionToBuffer(), and GDALWarpOperation::WarpRegionToBuffer().

Referenced by GDALWarpRegionToBuffer().