My Project

Contents

Chapter 1

GDAL Virtual Format Tutorial

1.1 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.

1.2 .vrt Format

A XML schema of the GDAL VRT format is available.

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

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

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

The allowed subelements for VRTDataset are :

- **SRS**: This element contains the spatial reference system (coordinate system) in OGC WKT format. Note that this must be appropriately escaped for XML, so items like quotes will have the ampersand escape sequences

substituted. As as well WKT, and valid input to the SetFromUserInput() method (such as well known GEO↩ GCS 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_Datum_1927
      &quot;,SPHEROID[&quot;Clarke 1866&quot;,6378206.4,294.9786982139006,AUTHORITY[&quot;EPSG&quot;,&quot;7008&
      quot;]],AUTHORITY[&quot;EPSG&quot;,&quot;6267&quot;]],PRIMEM[&quot;Greenwich&quot;,0],UNIT[&quot;degree&quot;,
      0.0174532925199433],AUTHORITY[&quot;EPSG&quot;,&quot;4267&quot;]],PROJECTION[&quot;Transverse_Mercator&quot;
      ],PARAMETER[&quot;latitude_of_origin&quot;,0],PARAMETER[&quot;central_meridian&quot;,-117],PARAMETER[&quot;
      scale_factor&quot;,0.9996],PARAMETER[&quot;false_easting&quot;,500000],PARAMETER[&quot;false_northing&quot;,0
      ],UNIT[&quot;metre&quot;,1,AUTHORITY[&quot;EPSG&quot;,&quot;9001&quot;]],AUTHORITY[&quot;EPSG&quot;,&quot;
      26711&quot;]]</SRS>
```

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

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

- **GCPList**: This element contains a list of Ground Control Points for the dataset, mapping between pixel/line coordinates and georeferenced coordinates. The Projection attribute should contain the SRS of the georeferenced coordinates in the same format as the SRS element.

```
<GCPList Projection="EPSG:4326">
  <GCP Id="1" Info="a" Pixel="0.5" Line="0.5" X="0.0" Y="0.0" Z="0.0" />
  <GCP Id="2" Info="b" Pixel="13.5" Line="23.5" X="1.0" Y="2.0" Z="0.0" />
</GCPList>
```

- **Metadata**: This element contains a list of metadata name/value pairs associated with the VRTDataset as a whole, or a VRTRasterBand. It has <MDI> (metadata item) subelements which have a "key" attribute and the value as the data of the element. The Metadata element can be repeated multiple times, in which case it must be accompanied with a "domain" attribute to indicate the name of the metadata domain.

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

- **MaskBand**: (GDAL >= 1.8.0) This element represents a mask band that is shared between all bands on the dataset (see GMF_PER_DATASET in RFC 15). It must contain a single VRTRasterBand child element, that is the description of the mask band itself.

```
<MaskBand>
  <VRTRasterBand dataType="Byte">
    <SimpleSource>
      <SourceFilename relativeToVRT="1">utm.tif</SourceFilename>
      <SourceBand>mask,1</SourceBand>
      <SrcRect xOff="0" yOff="0" xSize="512" ySize="512"/>
      <DstRect xOff="0" yOff="0" xSize="512" ySize="512"/>
    </SimpleSource>
  </VRTRasterBand>
</MaskBand>
```

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

  The allowed subelements for VRTRasterBand are :

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

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

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

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

– **HideNoDataValue**: If this value is 1, the nodata value will not be reported. Essentially, the caller will not be aware of a nodata pixel when it reads one. Any datasets copied/translated from this will not have a nodata value. This is useful when you want to specify a fixed background value for the dataset. The background will be the value specified by the NoDataValue element.

  Default value is 0 when this element is absent.

```
<HideNoDataValue>1</HideNoDataValue>
```

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

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

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

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

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

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

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

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

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

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

– **Overview**: This optional element describes one overview level for the band. It should have a child SourceFilename and SourceBand element. The SourceFilename may have a relativeToVRT boolean attribute. Multiple elements may be used to describe multiple overviews.

```
<Overview>
  <SourceFilename relativeToVRT="1">yellowstone_2.1.ntf.r2</SourceFilename>
  <SourceBand>1</SourceBand>
</Overview>
```

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

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

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

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

  The shared attribute, added in GDAL 2.0.0, on the SourceFilename indicates whether the dataset should be shared (value is 1) or not (value is 0). The default is 1. If several VRT datasets refering to the same underlying sources are used in a multithreaded context, shared should be set to 0. Alternatively, the VRT_SHARED_SOURCE configuration option can be set to 0 to force non-shared mode.

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

Starting with GDAL 1.8.0, the content of the SourceBand subelement can refer to a mask band. For example mask,1 means the the mask band of the first band of the source.

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

Starting with GDAL 2.0, a OpenOptions subelement can be added to specify the open options to apply when opening the source dataset. It has <OOI> (open option item) subelements which have a "key" attribute and the value as the data of the element.

```
<SimpleSource>
  <SourceFilename relativeToVRT="1">utm.tif</SourceFilename>
  <OpenOptions>
      <OOI key="OVERVIEW_LEVEL">0</OOI>
  </OpenOptions>
  <SourceBand>1</SourceBand>
  <SourceProperties RasterXSize="256" RasterYSize="256" DataType="Byte" BlockXSize="128" BlockYSize="128"/>
  <SrcRect xOff="0" yOff="0" xSize="256" ySize="256"/>
  <DstRect xOff="0" yOff="0" xSize="256" ySize="256"/>
</SimpleSource>
```

Starting with GDAL 2.0, a resampling attribute can be specified on a SimpleSource or Complex↩ Source element to specified the resampling algorithm used when the size of the destination rectangle is not the same as the size of the source rectangle. The values allowed for that attribute are : nearest,bilinear,cubic, cubicspline,lanczos,average,mode.

```
<SimpleSource resampling="cubic">
  <SourceFilename relativeToVRT="1">utm.tif</SourceFilename>
  <SourceBand>1</SourceBand>
  <SourceProperties RasterXSize="256" RasterYSize="256" DataType="Byte" BlockXSize="128" BlockYSize="128"/>
  <SrcRect xOff="0" yOff="0" xSize="256" ySize="256"/>
  <DstRect xOff="0" yOff="0" xSize="128" ySize="128"/>
</SimpleSource>
```

- **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 Simple↩ Source, when the size of the destination rectangle is not the same as the size of the source rectangle. Note: starting with GDAL 2.0, a more general mechanism to specify resampling algorithms can be used. See above paragraph about the 'resampling' attribute.
- **ComplexSource**: The ComplexSource is derived from the SimpleSource (so it shares the Source↩ Filename, 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.

  Starting with GDAL 1.11, alternatively to linear scaling, non-linear scaling using a power function can be used by specifying the Exponent, SrcMin, SrcMax, DstMin and DstMax elements. If SrcMin and Src↩ Max are not specified, they are computed from the source minimum and maximum value (which might require analyzing the whole source dataset). Exponent must be positive. (Those 5 values can be set with the -exponent and -scale options of gdal_translate.)

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

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

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

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

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

1. Nodata masking
2. Color table expansion
3. For linear scaling, applying the scale ratio, then scale offset
4. For non-linear scaling, apply (DstMax-DstMin) ∗ pow((SrcValue-SrcMin) / (SrcMax-SrcMin), Exponent) + DstMin
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>
```

Non-linear scaling:

```
<ComplexSource>
  <SourceFilename relativeToVRT="1">16bit.tif</SourceFilename>
  <SourceBand>1</SourceBand>
  <Exponent>0.75</Exponent>
  <SrcMin>0</SrcMin>
  <SrcMax>65535</SrcMax>
  <DstMin>0</DstMin>
  <DstMax>255</DstMax>
  <SrcRect xOff="0" yOff="0" xSize="512" ySize="512"/>
  <DstRect xOff="0" yOff="0" xSize="512" ySize="512"/>
</ComplexSource>
```

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

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

– **MaskBand**: (GDAL >= 1.8.0) This element represents a mask band that is specific to the VRTRasterBand it contains. It must contain a single VRTRasterBand child element, that is the description of the mask band itself.

1.3 .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 VRTRawRasterBand contains a number of previously unseen elements but no "source" information. VRTRaw↩ RasterBands 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 VRTRaw↩ RasterBand.
- 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, Pixel↩ Offset, 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>
```

1.4 Programatic Creation of VRT Datasets

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

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

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

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

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

GDALClose((GDALDatasetH) poVRTDS);
GDALClose((GDALDatasetH) poSrcDS);
```

Note the use of GDALOpenShared() when opening the source dataset. It is advised to use GDALOpenShared() in this situation so that you are able to release the explicit reference to it before closing the VRT dataset itself. In other words, in the previous example, you could also invert the 2 last lines, whereas if you open the source dataset with GDALOpen(), you'd need to close the VRT dataset before closing the source dataset.

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

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

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

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

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

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

GDALClose((GDALDatasetH) poTiffDS);
```

In the above example the nodata value is set as -999. You can set the HideNoDataValue element in the VRT dataset's band using SetMetadataItem() on that band.

```
poVRTDS->GetRasterBand( 1 )->SetMetadataItem( "HideNoDataValue" , "1" );
```

In this example a virtual dataset is created with the Create() method, and adding bands and sources programmatically, but still via the "generic" API. A special attribute of VRT datasets is that sources can be added to the VRTRasterBand (but not to VRTRawRasterBand) by passing the XML describing the source into SetMetadata() on the special domain target "new_vrt_sources". The domain target "vrt_sources" may also be used, in which case any existing sources will be discarded before adding the new ones. In this example we construct a simple averaging filter source instead of using the simple source.

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

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

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

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

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

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

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

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

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

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

The VRTDataset class is one of the few dataset implementations that supports the AddBand() method. The options passed to the AddBand() method can be used to control the type of the band created (VRTRasterBand, VRTRaw↩RasterBand, VRTDerivedRasterBand), and in the case of the VRTRawRasterBand to set its various parameters. For standard VRTRasterBand, sources should be specified with the above SetMetadata() / SetMetadataItem() examples.

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

poVRTDS = poDriver->Create( "out.vrt", 512, 512, 0, GDT_Byte, NULL );
char** papszOptions = NULL;
papszOptions = CSLAddNameValue(papszOptions, "subclass", "VRTRawRasterBand"); // if not specified, default
       to VRTRasterBand
papszOptions = CSLAddNameValue(papszOptions, "SourceFilename", "src.tif"); // mandatory
papszOptions = CSLAddNameValue(papszOptions, "ImageOffset", "156"); // optionnal. default = 0
papszOptions = CSLAddNameValue(papszOptions, "PixelOffset", "2"); // optionnal. default = size of band type

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

delete poVRTDS;
```

Using Derived Bands

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

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

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

You could write the pixel function to compute this value and then register it with GDAL with the name "MyFirst↩Function". 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;
}
```

1.5 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.

The shared attribute, added in GDAL 2.0.0, on the SourceFilename indicates whether the dataset should be shared (value is 1) or not (value is 0). The default is 1. If several VRT datasets refering to the same underlying sources are used in a multithreaded context, shared should be set to 0. Alternatively, the VRT_SHARED_SOURCE configuration option can be set to 0 to force non-shared mode.

1.6 Performance considerations

A VRT can reference many (hundreds, thousands, or more) datasets. Due to operating system limitations, and for performance at opening time, it is not reasonable/possible to open them all at the same time. GDAL has a "pool" of datasets opened by VRT files whose maximum limit is 100 by default. When it needs to access a dataset referenced by a VRT, it checks if it is already in the pool of open datasets. If not, when the pool has reached its limit, it closes the least recently used dataset to be able to open the new one. This maximum limit of the pool can be increased by setting the GDAL_MAX_DATASET_POOL_SIZE configuration option to a bigger value. Note that a typical user process on Linux is limited to 1024 simultaneously opened files, and you should let some margin for shared libraries, etc... As of GDAL 2.0, gdal_translate and gdalwarp, by default, increase the pool size to 450.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

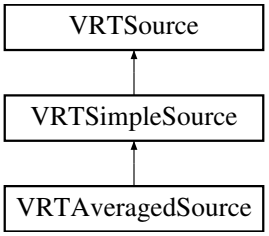
Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 VRTAveragedSource Class Reference

Inheritance diagram for VRTAveragedSource:



Public Member Functions

- virtual CPLErr **RasterIO** (int nXOff, int nYOff, int nXSize, int nYSize, void ∗pData, int nBufXSize, int nBuf↩ YSize, GDALDataType eBufType, GSpacing nPixelSpace, GSpacing nLineSpace, GDALRasterIOExtraArg ∗psExtraArg)
- virtual double **GetMinimum** (int nXSize, int nYSize, int ∗pbSuccess)
- virtual double **GetMaximum** (int nXSize, int nYSize, int ∗pbSuccess)
- virtual CPLErr **ComputeRasterMinMax** (int nXSize, int nYSize, int bApproxOK, double ∗adfMinMax)
- virtual CPLErr **ComputeStatistics** (int nXSize, int nYSize, int bApproxOK, double ∗pdfMin, double ∗pdfMax, double ∗pdfMean, double ∗pdfStdDev, GDALProgressFunc pfnProgress, void ∗pProgressData)
- virtual CPLErr **GetHistogram** (int nXSize, int nYSize, double dfMin, double dfMax, int nBuckets, GUInt↩ Big ∗panHistogram, int bIncludeOutOfRange, int bApproxOK, GDALProgressFunc pfnProgress, void ∗p↩ ProgressData)
- virtual CPLXMLNode ∗ **SerializeToXML** (const char ∗pszVRTPath)
- virtual const char ∗ **GetType** ()

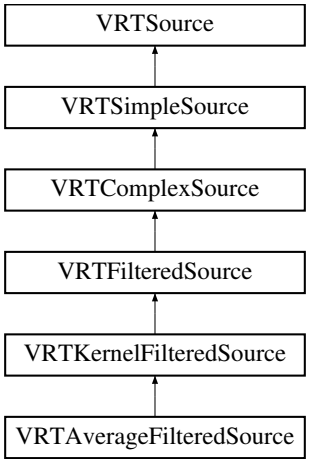
Additional Inherited Members

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

- vrtdataset.h
- vrtsources.cpp

5.2 VRTAverageFilteredSource Class Reference

Inheritance diagram for VRTAverageFilteredSource:



Public Member Functions

- **VRTAverageFilteredSource** (int nKernelSize)
- virtual CPLErr **XMLInit** (CPLXMLNode ∗psTree, const char ∗)
- virtual CPLXMLNode ∗ **SerializeToXML** (const char ∗pszVRTPath)

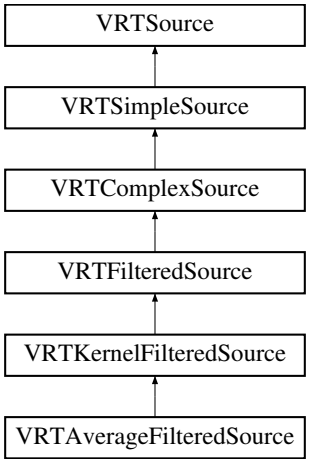
Additional Inherited Members

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

- vrtdataset.h

5.3 VRTComplexSource Class Reference

Inheritance diagram for VRTComplexSource:



Public Member Functions

- virtual CPLErr **RasterIO** (int nXOff, int nYOff, int nXSize, int nYSize, void ∗pData, int nBufXSize, int nBuf↩ YSize, GDALDataType eBufType, GSpacing nPixelSpace, GSpacing nLineSpace, GDALRasterIOExtraArg ∗psExtraArg)
- virtual double **GetMinimum** (int nXSize, int nYSize, int ∗pbSuccess)
- virtual double **GetMaximum** (int nXSize, int nYSize, int ∗pbSuccess)
- virtual CPLErr **ComputeRasterMinMax** (int nXSize, int nYSize, int bApproxOK, double ∗adfMinMax)
- virtual CPLErr **ComputeStatistics** (int nXSize, int nYSize, int bApproxOK, double ∗pdfMin, double ∗pdfMax, double ∗pdfMean, double ∗pdfStdDev, GDALProgressFunc pfnProgress, void ∗pProgressData)
- virtual CPLErr **GetHistogram** (int nXSize, int nYSize, double dfMin, double dfMax, int nBuckets, GUInt↩ Big ∗panHistogram, int bIncludeOutOfRange, int bApproxOK, GDALProgressFunc pfnProgress, void ∗p↩ ProgressData)
- virtual CPLXMLNode ∗ **SerializeToXML** (const char ∗pszVRTPath)
- virtual CPLErr **XMLInit** (CPLXMLNode ∗, const char ∗)
- virtual const char ∗ **GetType** ()
- double **LookupValue** (double dfInput)
- void **SetLinearScaling** (double dfOffset, double dfScale)
- void **SetPowerScaling** (double dfExponent, double dfSrcMin, double dfSrcMax, double dfDstMin, double dfDstMax)
- void **SetColorTableComponent** (int nComponent)

Public Attributes

- double ∗ **padfLUTInputs**
- double ∗ **padfLUTOutputs**
- int **nLUTItemCount**

Protected Member Functions

- CPLErr **RasterIOInternal** (int nReqXOff, int nReqYOff, int nReqXSize, int nReqYSize, void ∗pData, int n↩ OutXSize, int nOutYSize, GDALDataType eBufType, GSpacing nPixelSpace, GSpacing nLineSpace, GDA↩ LRasterIOExtraArg ∗psExtraArg)

Protected Attributes

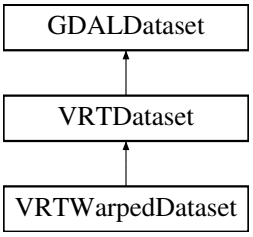
- VRTComplexSourceScaling **eScalingType**
- double **dfScaleOff**
- double **dfScaleRatio**
- int **bSrcMinMaxDefined**
- double **dfSrcMin**
- double **dfSrcMax**
- double **dfDstMin**
- double **dfDstMax**
- double **dfExponent**
- int **nColorTableComponent**

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

- vrtdataset.h
- vrtsources.cpp

5.4 VRTDataset Class Reference

Inheritance diagram for VRTDataset:



Public Member Functions

- **VRTDataset** (int nXSize, int nYSize)
- void **SetNeedsFlush** ()
- virtual void **FlushCache** ()
- void **SetWritable** (int bWritable)
- virtual CPLErr **CreateMaskBand** (int nFlags)
- void **SetMaskBand** (VRTRasterBand *poMaskBand)
- virtual const char * **GetProjectionRef** (void)
- virtual CPLErr **SetProjection** (const char *)
- virtual CPLErr **GetGeoTransform** (double *)
- virtual CPLErr **SetGeoTransform** (double *)
- virtual CPLErr **SetMetadata** (char **papszMD, const char *pszDomain="")
- virtual CPLErr **SetMetadataItem** (const char *pszName, const char *pszValue, const char *pszDomain="")
- 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 **AddBand** (GDALDataType eType, char **papszOptions=NULL)
- virtual char ** **GetFileList** ()
- virtual CPLErr **IRasterIO** (GDALRWFlag eRWFlag, int nXOff, int nYOff, int nXSize, int nYSize, void *p↩ Data, int nBufXSize, int nBufYSize, GDALDataType eBufType, int nBandCount, int *panBandMap, GSpacing nPixelSpace, GSpacing nLineSpace, GSpacing nBandSpace, GDALRasterIOExtraArg *psExtraArg)
- virtual CPLXMLNode * **SerializeToXML** (const char *pszVRTPath)
- virtual CPLErr **XMLInit** (CPLXMLNode *, const char *)
- GDALDataset * **GetSingleSimpleSource** ()
- void **UnsetPreservedRelativeFilenames** ()

Static Public Member Functions

- static int **Identify** (GDALOpenInfo *)
- static GDALDataset * **Open** (GDALOpenInfo *)
- static GDALDataset * **OpenXML** (const char *, const char *=NULL, GDALAccess eAccess=GA_ReadOnly)
- static GDALDataset * **Create** (const char *pszName, int nXSize, int nYSize, int nBands, GDALDataType eType, char **papszOptions)
- static CPLErr **Delete** (const char *pszFilename)

Protected Member Functions

- virtual int **CloseDependentDatasets** ()

Friends

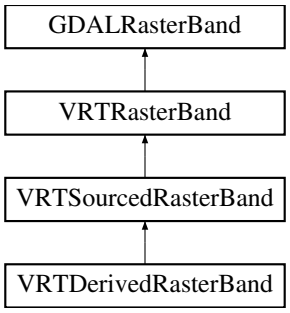
- class **VRTRasterBand**

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

- vrtdataset.h
- vrtdataset.cpp

5.5 VRTDerivedRasterBand Class Reference

Inheritance diagram for VRTDerivedRasterBand:



Public Member Functions

- **VRTDerivedRasterBand** (GDALDataset ∗poDS, int nBand)
- **VRTDerivedRasterBand** (GDALDataset ∗poDS, int nBand, GDALDataType eType, int nXSize, int nYSize)
- virtual CPLErr IRasterIO (GDALRWFlag, int, int, int, int, void ∗, int, int, GDALDataType, GSpacing nPixel↩ Space, GSpacing nLineSpace, GDALRasterIOExtraArg ∗psExtraArg)
- void SetPixelFunctionName (const char ∗pszFuncName)
- void SetSourceTransferType (GDALDataType eDataType)
- virtual CPLErr **XMLInit** (CPLXMLNode ∗, const char ∗)
- virtual CPLXMLNode ∗ **SerializeToXML** (const char ∗pszVRTPath)

Static Public Member Functions

- static CPLErr AddPixelFunction (const char ∗pszFuncName, GDALDerivedPixelFunc pfnPixelFunc)
- static GDALDerivedPixelFunc GetPixelFunction (const char ∗pszFuncName)

Public Attributes

- char ∗ **pszFuncName**
- GDALDataType **eSourceTransferType**

Additional Inherited Members

5.5.1 Member Function Documentation

5.5.1.1 CPLErr VRTDerivedRasterBand::AddPixelFunction (const char * *pszFuncName*, GDALDerivedPixelFunc *pfnNewFunction*) [static]

This adds a pixel function to the global list of available pixel functions for derived bands.

This is the same as the c function GDALAddDerivedBandPixelFunc()

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.

5.5.1.2 GDALDerivedPixelFunc VRTDerivedRasterBand::GetPixelFunction (const char * *pszFuncName*) [static]

Get a pixel function previously registered using the global AddPixelFunction.

Parameters

pszFuncName	The name associated with the pixel function.

Returns

A derived band pixel function, or NULL if none have been registered for pszFuncName.

5.5.1.3 CPLErr VRTDerivedRasterBand::IRasterIO (GDALRWFlag *eRWFlag,* int *nXOff,* int *nYOff,* int *nXSize,* int *nYSize,* void * *pData,* int *nBufXSize,* int *nBufYSize,* GDALDataType *eBufType,* GSpacing *nPixelSpace,* GSpacing *nLineSpace,* GDALRasterIOExtraArg * *psExtraArg*) [virtual]

Read/write a region of image data for this band.

Each of the sources for this derived band will be read and passed to the derived band pixel function. The pixel function is responsible for applying whatever algorithm is necessary to generate this band's pixels from the sources.

The sources will be read using the transfer type specified for sources using SetSourceTransferType(). If no transfer type has been set for this derived band, the band's data type will be used as the transfer type.

See also

gdalrasterband

Parameters

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

Reimplemented from VRTSourcedRasterBand.

5.5.1.4 void VRTDerivedRasterBand::SetPixelFunctionName (const char * *pszFuncName*)

Set the pixel function name to be applied to this derived band. The name should match a pixel function registered using AddPixelFunction.

Parameters

pszFuncName	Name of pixel function to be applied to this derived band.

5.5.1.5 void VRTDerivedRasterBand::SetSourceTransferType (GDALDataType *eDataType*)

Set the transfer type to be used to obtain pixel information from all of the sources. If unset, the transfer type used will be the same as the derived band data type. This makes it possible, for example, to pass CFloat32 source pixels to the pixel function, even if the pixel function generates a raster for a derived band that is of type Byte.

Parameters

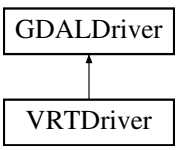
eDataType	Data type to use to obtain pixel information from the sources to be passed to the derived band pixel function.

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

- vrtdataset.h
- vrtderivedrasterband.cpp

5.6 VRTDriver Class Reference

Inheritance diagram for VRTDriver:



Public Member Functions

- virtual char ** **GetMetadataDomainList** ()

- virtual char ** **GetMetadata** (const char *pszDomain="")
- virtual CPLErr **SetMetadata** (char **papszMetadata, const char *pszDomain="")
- VRTSource * **ParseSource** (CPLXMLNode *psSrc, const char *pszVRTPath)
- void **AddSourceParser** (const char *pszElementName, VRTSourceParser pfnParser)

Public Attributes

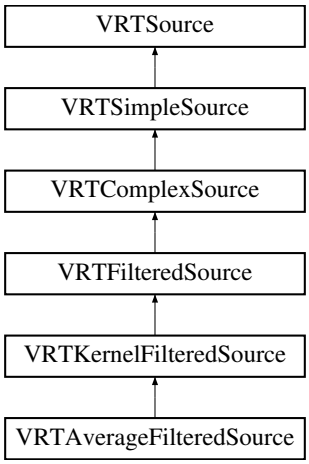
- char ** **papszSourceParsers**

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

- vrtdataset.h
- vrtdriver.cpp

5.7 VRTFilteredSource Class Reference

Inheritance diagram for VRTFilteredSource:



Public Member Functions

- void **SetExtraEdgePixels** (int)
- void **SetFilteringDataTypesSupported** (int, GDALDataType *)
- virtual CPLErr **FilterData** (int nXSize, int nYSize, GDALDataType eType, GByte *pabySrcData, GByte *pabyDstData)=0
- virtual CPLErr **RasterIO** (int nXOff, int nYOff, int nXSize, int nYSize, void *pData, int nBufXSize, int nBuf↩ YSize, GDALDataType eBufType, GSpacing nPixelSpace, GSpacing nLineSpace, GDALRasterIOExtraArg *psExtraArg)

Protected Attributes

- int **nSupportedTypesCount**
- GDALDataType **aeSupportedTypes** [20]
- int **nExtraEdgePixels**

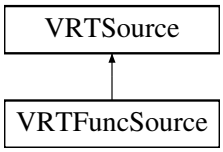
Additional Inherited Members

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

- vrtdataset.h
- vrtfilters.cpp

5.8 VRTFuncSource Class Reference

Inheritance diagram for VRTFuncSource:



Public Member Functions

- virtual CPLErr **XMLInit** (CPLXMLNode ∗, const char ∗)
- virtual CPLXMLNode ∗ **SerializeToXML** (const char ∗pszVRTPath)
- virtual CPLErr **RasterIO** (int nXOff, int nYOff, int nXSize, int nYSize, void ∗pData, int nBufXSize, int nBuf↩ YSize, GDALDataType eBufType, GSpacing nPixelSpace, GSpacing nLineSpace, GDALRasterIOExtraArg ∗psExtraArg)
- virtual double **GetMinimum** (int nXSize, int nYSize, int ∗pbSuccess)
- virtual double **GetMaximum** (int nXSize, int nYSize, int ∗pbSuccess)
- virtual CPLErr **ComputeRasterMinMax** (int nXSize, int nYSize, int bApproxOK, double ∗adfMinMax)
- virtual CPLErr **ComputeStatistics** (int nXSize, int nYSize, int bApproxOK, double ∗pdfMin, double ∗pdfMax, double ∗pdfMean, double ∗pdfStdDev, GDALProgressFunc pfnProgress, void ∗pProgressData)
- virtual CPLErr **GetHistogram** (int nXSize, int nYSize, double dfMin, double dfMax, int nBuckets, GUInt↩ Big ∗panHistogram, int bIncludeOutOfRange, int bApproxOK, GDALProgressFunc pfnProgress, void ∗p↩ ProgressData)

Public Attributes

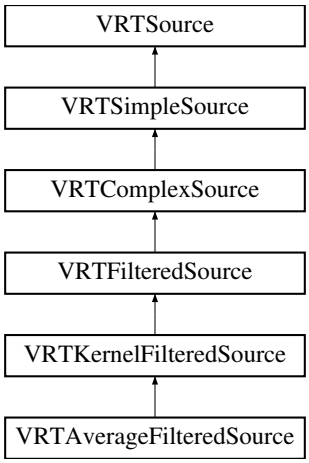
- VRTImageReadFunc **pfnReadFunc**
- void ∗ **pCBData**
- GDALDataType **eType**
- float **fNoDataValue**

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

- vrtdataset.h
- vrtsources.cpp

5.9 VRTKernelFilteredSource Class Reference

Inheritance diagram for VRTKernelFilteredSource:



Public Member Functions

- virtual CPLErr **XMLInit** (CPLXMLNode ∗psTree, const char ∗)
- virtual CPLXMLNode ∗ **SerializeToXML** (const char ∗pszVRTPath)
- virtual CPLErr **FilterData** (int nXSize, int nYSize, GDALDataType eType, GByte ∗pabySrcData, GByte ∗pabyDstData)
- CPLErr **SetKernel** (int nKernelSize, double ∗padfCoefs)
- void **SetNormalized** (int)

Protected Attributes

- int **nKernelSize**
- double ∗ **padfKernelCoefs**
- int **bNormalized**

Additional Inherited Members

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

- vrtdataset.h
- vrtfilters.cpp

5.10 VRTOverviewInfo Class Reference

Public Attributes

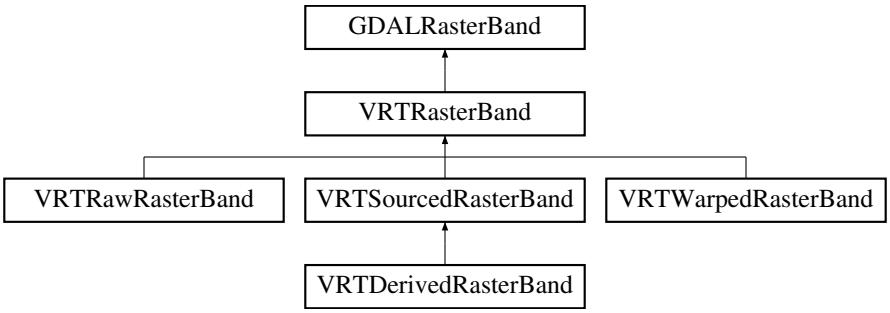
- CPLString **osFilename**
- int **nBand**
- GDALRasterBand ∗ **poBand**
- int **bTriedToOpen**

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

- vrtdataset.h

5.11 VRTRasterBand Class Reference

Inheritance diagram for VRTRasterBand:



Public Member Functions

- virtual CPLErr **XMLInit** (CPLXMLNode *, const char *)
- virtual CPLXMLNode * **SerializeToXML** (const char *pszVRTPath)
- virtual CPLErr **SetNoDataValue** (double)
- virtual double **GetNoDataValue** (int *pbSuccess=NULL)
- virtual CPLErr **SetColorTable** (GDALColorTable *)
- virtual GDALColorTable * **GetColorTable** ()
- virtual CPLErr **SetColorInterpretation** (GDALColorInterp)
- virtual GDALColorInterp **GetColorInterpretation** ()
- virtual const char * **GetUnitType** ()
- CPLErr **SetUnitType** (const char *)
- virtual char ** **GetCategoryNames** ()
- virtual CPLErr **SetCategoryNames** (char **)
- virtual CPLErr **SetMetadata** (char **papszMD, const char *pszDomain="")
- virtual CPLErr **SetMetadataItem** (const char *pszName, const char *pszValue, const char *pszDomain="")
- virtual double **GetOffset** (int *pbSuccess=NULL)
- CPLErr **SetOffset** (double)
- virtual double **GetScale** (int *pbSuccess=NULL)
- CPLErr **SetScale** (double)
- virtual int **GetOverviewCount** ()
- virtual GDALRasterBand * **GetOverview** (int)
- virtual CPLErr **GetHistogram** (double dfMin, double dfMax, int nBuckets, GUIntBig *panHistogram, int b↩ IncludeOutOfRange, int bApproxOK, GDALProgressFunc, void *pProgressData)
- virtual CPLErr **GetDefaultHistogram** (double *pdfMin, double *pdfMax, int *pnBuckets, GUIntBig **ppan↩ Histogram, int bForce, GDALProgressFunc, void *pProgressData)
- virtual CPLErr **SetDefaultHistogram** (double dfMin, double dfMax, int nBuckets, GUIntBig *panHistogram)
- CPLErr **CopyCommonInfoFrom** (GDALRasterBand *)
- virtual void **GetFileList** (char ***ppapszFileList, int *pnSize, int *pnMaxSize, CPLHashSet *hSetFiles)
- virtual void **SetDescription** (const char *)
- virtual GDALRasterBand * **GetMaskBand** ()
- virtual int **GetMaskFlags** ()
- virtual CPLErr **CreateMaskBand** (int nFlags)
- void **SetMaskBand** (VRTRasterBand *poMaskBand)
- void **SetIsMaskBand** ()
- CPLErr **UnsetNoDataValue** ()
- virtual int **CloseDependentDatasets** ()
- virtual int **IsSourcedRasterBand** ()

Protected Member Functions

- void **Initialize** (int nXSize, int nYSize)

Protected Attributes

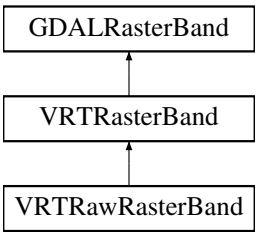
- int **bIsMaskBand**
- int **bNoDataValueSet**
- int **bHideNoDataValue**
- double **dfNoDataValue**
- GDALColorTable * **poColorTable**
- GDALColorInterp **eColorInterp**
- char * **pszUnitType**
- char ** **papszCategoryNames**
- double **dfOffset**
- double **dfScale**
- CPLXMLNode * **psSavedHistograms**
- std::vector< VRTOverviewInfo > **apoOverviews**
- VRTRasterBand * **poMaskBand**

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

- vrtdataset.h
- vrtrasterband.cpp

5.12 VRTRawRasterBand Class Reference

Inheritance diagram for VRTRawRasterBand:



Public Member Functions

- **VRTRawRasterBand** (GDALDataset *poDS, int nBand, GDALDataType eType=GDT_Unknown)
- virtual CPLErr **XMLInit** (CPLXMLNode *, const char *)
- virtual CPLXMLNode * **SerializeToXML** (const char *pszVRTPath)
- virtual CPLErr **IRasterIO** (GDALRWFlag, int, int, int, int, void *, int, int, GDALDataType, GSpacing nPixel↩ Space, GSpacing nLineSpace, GDALRasterIOExtraArg *psExtraArg)
- virtual CPLErr **IReadBlock** (int, int, void *)
- virtual CPLErr **IWriteBlock** (int, int, void *)
- CPLErr **SetRawLink** (const char *pszFilename, const char *pszVRTPath, int bRelativeToVRT, vsi_l_offset nImageOffset, int nPixelOffset, int nLineOffset, const char *pszByteOrder)
- void **ClearRawLink** ()
- virtual void **GetFileList** (char ***ppapszFileList, int *pnSize, int *pnMaxSize, CPLHashSet *hSetFiles)

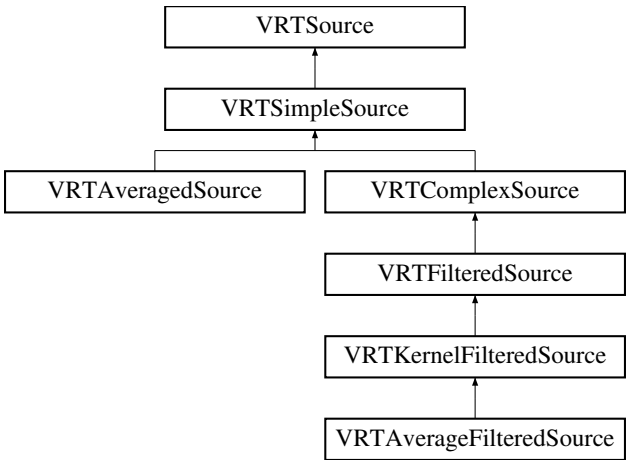
Additional Inherited Members

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

- vrtdataset.h
- vrtrawrasterband.cpp

5.13 VRTSimpleSource Class Reference

Inheritance diagram for VRTSimpleSource:



Public Member Functions

- virtual CPLErr **XMLInit** (CPLXMLNode ∗psTree, const char ∗)
- virtual CPLXMLNode ∗ **SerializeToXML** (const char ∗pszVRTPath)
- void **SetSrcBand** (GDALRasterBand ∗)
- void **SetSrcMaskBand** (GDALRasterBand ∗)
- void **SetSrcWindow** (int, int, int, int)
- void **SetDstWindow** (int, int, int, int)
- void **SetNoDataValue** (double dfNoDataValue)
- const CPLString & **GetResampling** () const
- void **SetResampling** (const char ∗pszResampling)
- int **GetSrcDstWindow** (int, int, int, int, int, int, double ∗pdfReqXOff, double ∗pdfReqYOff, double ∗pdfReq↩ XSize, double ∗pdfReqYSize, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗, int ∗)
- virtual CPLErr **RasterIO** (int nXOff, int nYOff, int nXSize, int nYSize, void ∗pData, int nBufXSize, int nBuf↩ YSize, GDALDataType eBufType, GSpacing nPixelSpace, GSpacing nLineSpace, GDALRasterIOExtraArg ∗psExtraArg)
- virtual double **GetMinimum** (int nXSize, int nYSize, int ∗pbSuccess)
- virtual double **GetMaximum** (int nXSize, int nYSize, int ∗pbSuccess)
- virtual CPLErr **ComputeRasterMinMax** (int nXSize, int nYSize, int bApproxOK, double ∗adfMinMax)
- virtual CPLErr **ComputeStatistics** (int nXSize, int nYSize, int bApproxOK, double ∗pdfMin, double ∗pdfMax, double ∗pdfMean, double ∗pdfStdDev, GDALProgressFunc pfnProgress, void ∗pProgressData)
- virtual CPLErr **GetHistogram** (int nXSize, int nYSize, double dfMin, double dfMax, int nBuckets, GUInt↩ Big ∗panHistogram, int bIncludeOutOfRange, int bApproxOK, GDALProgressFunc pfnProgress, void ∗p↩ ProgressData)
- void **DstToSrc** (double dfX, double dfY, double &dfXOut, double &dfYOut)
- void **SrcToDst** (double dfX, double dfY, double &dfXOut, double &dfYOut)
- virtual void **GetFileList** (char ∗∗∗ppapszFileList, int ∗pnSize, int ∗pnMaxSize, CPLHashSet ∗hSetFiles)

- virtual int **IsSimpleSource** ()
- virtual const char * **GetType** ()
- GDALRasterBand * **GetBand** ()
- int **IsSameExceptBandNumber** (VRTSimpleSource *poOtherSource)
- CPLErr **DatasetRasterIO** (int nXOff, int nYOff, int nXSize, int nYSize, void *pData, int nBufXSize, int nBufYSize, GDALDataType eBufType, int nBandCount, int *panBandMap, GSpacing nPixelSpace, GSpacing nLineSpace, GSpacing nBandSpace, GDALRasterIOExtraArg *psExtraArg)
- void **UnsetPreservedRelativeFilenames** ()

Protected Attributes

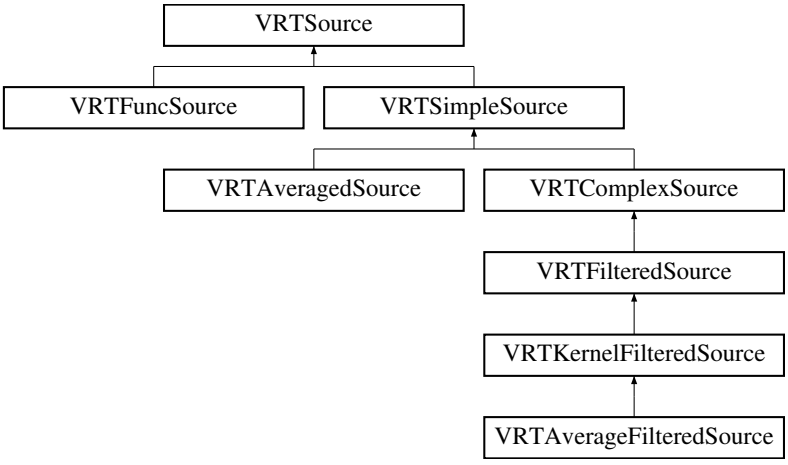
- GDALRasterBand * **poRasterBand**
- GDALRasterBand * **poMaskBandMainBand**
- int **nSrcXOff**
- int **nSrcYOff**
- int **nSrcXSize**
- int **nSrcYSize**
- int **nDstXOff**
- int **nDstYOff**
- int **nDstXSize**
- int **nDstYSize**
- int **bNoDataSet**
- double **dfNoDataValue**
- CPLString **osResampling**
- int **bRelativeToVRTOri**
- CPLString **osSourceFileNameOri**

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

- vrtdataset.h
- vrtsources.cpp

5.14 VRTSource Class Reference

Inheritance diagram for VRTSource:



Public Member Functions

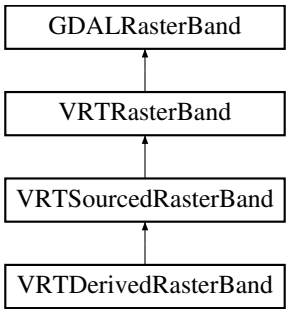
- virtual CPLErr **RasterIO** (int nXOff, int nYOff, int nXSize, int nYSize, void ∗pData, int nBufXSize, int nBuf↩ YSize, GDALDataType eBufType, GSpacing nPixelSpace, GSpacing nLineSpace, GDALRasterIOExtraArg ∗psExtraArg)=0
- virtual double **GetMinimum** (int nXSize, int nYSize, int ∗pbSuccess)=0
- virtual double **GetMaximum** (int nXSize, int nYSize, int ∗pbSuccess)=0
- virtual CPLErr **ComputeRasterMinMax** (int nXSize, int nYSize, int bApproxOK, double ∗adfMinMax)=0
- virtual CPLErr **ComputeStatistics** (int nXSize, int nYSize, int bApproxOK, double ∗pdfMin, double ∗pdfMax, double ∗pdfMean, double ∗pdfStdDev, GDALProgressFunc pfnProgress, void ∗pProgressData)=0
- virtual CPLErr **GetHistogram** (int nXSize, int nYSize, double dfMin, double dfMax, int nBuckets, GUInt↩ Big ∗panHistogram, int bIncludeOutOfRange, int bApproxOK, GDALProgressFunc pfnProgress, void ∗p↩ ProgressData)=0
- virtual CPLErr **XMLInit** (CPLXMLNode ∗psTree, const char ∗)=0
- virtual CPLXMLNode ∗ **SerializeToXML** (const char ∗pszVRTPath)=0
- virtual void **GetFileList** (char ∗∗∗ppapszFileList, int ∗pnSize, int ∗pnMaxSize, CPLHashSet ∗hSetFiles)
- virtual int **IsSimpleSource** ()

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

- vrtdataset.h
- vrtsources.cpp

5.15 VRTSourcedRasterBand Class Reference

Inheritance diagram for VRTSourcedRasterBand:



Public Member Functions

- **VRTSourcedRasterBand** (GDALDataset ∗poDS, int nBand)
- **VRTSourcedRasterBand** (GDALDataType eType, int nXSize, int nYSize)
- **VRTSourcedRasterBand** (GDALDataset ∗poDS, int nBand, GDALDataType eType, int nXSize, int nYSize)
- virtual CPLErr **IRasterIO** (GDALRWFlag, int, int, int, int, void ∗, int, int, GDALDataType, GSpacing nPixel↩ Space, GSpacing nLineSpace, GDALRasterIOExtraArg ∗psExtraArg)
- virtual char ∗∗ **GetMetadataDomainList** ()
- virtual const char ∗ **GetMetadataItem** (const char ∗pszName, const char ∗pszDomain="")
- virtual char ∗∗ **GetMetadata** (const char ∗pszDomain="")
- virtual CPLErr **SetMetadata** (char ∗∗papszMetadata, const char ∗pszDomain="")
- virtual CPLErr **SetMetadataItem** (const char ∗pszName, const char ∗pszValue, const char ∗pszDomain="")
- virtual CPLErr **XMLInit** (CPLXMLNode ∗, const char ∗)
- virtual CPLXMLNode ∗ **SerializeToXML** (const char ∗pszVRTPath)
- virtual double **GetMinimum** (int ∗pbSuccess=NULL)

- virtual double **GetMaximum** (int *pbSuccess=NULL)
- virtual CPLErr **ComputeRasterMinMax** (int bApproxOK, double *adfMinMax)
- virtual CPLErr **ComputeStatistics** (int bApproxOK, double *pdfMin, double *pdfMax, double *pdfMean, double *pdfStdDev, GDALProgressFunc pfnProgress, void *pProgressData)
- virtual CPLErr **GetHistogram** (double dfMin, double dfMax, int nBuckets, GUIntBig *panHistogram, int bIncludeOutOfRange, int bApproxOK, GDALProgressFunc pfnProgress, void *pProgressData)
- CPLErr **AddSource** (VRTSource *)
- CPLErr **AddSimpleSource** (GDALRasterBand *poSrcBand, int nSrcXOff=-1, int nSrcYOff=-1, int nSrcXSize=-1, int nSrcYSize=-1, int nDstXOff=-1, int nDstYOff=-1, int nDstXSize=-1, int nDstYSize=-1, const char *pszResampling="near", double dfNoDataValue=VRT_NODATA_UNSET)
- CPLErr **AddComplexSource** (GDALRasterBand *poSrcBand, int nSrcXOff=-1, int nSrcYOff=-1, int nSrcXSize=-1, int nSrcYSize=-1, int nDstXOff=-1, int nDstYOff=-1, int nDstXSize=-1, int nDstYSize=-1, double dfScaleOff=0.0, double dfScaleRatio=1.0, double dfNoDataValue=VRT_NODATA_UNSET, int nColorTableComponent=0)
- CPLErr **AddMaskBandSource** (GDALRasterBand *poSrcBand, int nSrcXOff=-1, int nSrcYOff=-1, int nSrcXSize=-1, int nSrcYSize=-1, int nDstXOff=-1, int nDstYOff=-1, int nDstXSize=-1, int nDstYSize=-1)
- CPLErr **AddFuncSource** (VRTImageReadFunc pfnReadFunc, void *hCBData, double dfNoDataValue=VRT_NODATA_UNSET)
- void **ConfigureSource** (VRTSimpleSource *poSimpleSource, GDALRasterBand *poSrcBand, int bAddAsMaskBand, int nSrcXOff, int nSrcYOff, int nSrcXSize, int nSrcYSize, int nDstXOff, int nDstYOff, int nDstXSize, int nDstYSize)
- virtual CPLErr **IReadBlock** (int, int, void *)
- virtual void **GetFileList** (char ***ppapszFileList, int *pnSize, int *pnMaxSize, CPLHashSet *hSetFiles)
- virtual int **CloseDependentDatasets** ()
- virtual int **IsSourcedRasterBand** ()

Public Attributes

- int **nSources**
- VRTSource ** **papoSources**
- int **bEqualAreas**

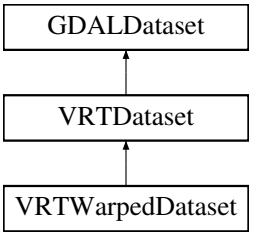
Additional Inherited Members

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

- vrtdataset.h
- vrtsourcedrasterband.cpp

5.16 VRTWarpedDataset Class Reference

Inheritance diagram for VRTWarpedDataset:



Public Member Functions

- **VRTWarpedDataset** (int nXSize, int nYSize)
- CPLErr **Initialize** (void ∗)
- virtual CPLErr **IBuildOverviews** (const char ∗, int, int ∗, int, int ∗, GDALProgressFunc, void ∗)
- virtual CPLErr **SetMetadataItem** (const char ∗pszName, const char ∗pszValue, const char ∗pszDomain="")
- virtual CPLXMLNode ∗ **SerializeToXML** (const char ∗pszVRTPath)
- virtual CPLErr **XMLInit** (CPLXMLNode ∗, const char ∗)
- virtual CPLErr **AddBand** (GDALDataType eType, char ∗∗papszOptions=NULL)
- virtual char ∗∗ **GetFileList** ()
- CPLErr **ProcessBlock** (int iBlockX, int iBlockY)
- void **GetBlockSize** (int ∗, int ∗)

Protected Member Functions

- virtual int **CloseDependentDatasets** ()

Friends

- class **VRTWarpedRasterBand**

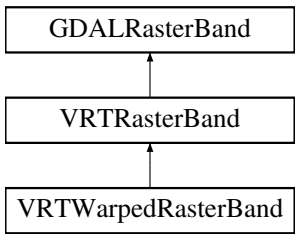
Additional Inherited Members

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

- vrtdataset.h
- vrtwarped.cpp

5.17 VRTWarpedRasterBand Class Reference

Inheritance diagram for VRTWarpedRasterBand:



Public Member Functions

- **VRTWarpedRasterBand** (GDALDataset ∗poDS, int nBand, GDALDataType eType=GDT_Unknown)
- virtual CPLErr **XMLInit** (CPLXMLNode ∗, const char ∗)
- virtual CPLXMLNode ∗ **SerializeToXML** (const char ∗pszVRTPath)
- virtual CPLErr **IReadBlock** (int, int, void ∗)
- virtual CPLErr **IWriteBlock** (int, int, void ∗)
- virtual int **GetOverviewCount** ()
- virtual GDALRasterBand ∗ **GetOverview** (int)

Additional Inherited Members

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

- vrtdataset.h
- vrtwarped.cpp

5.18 VWOTInfo Struct Reference

Public Attributes

- GDALTransformerInfo **sTI**
- GDALTransformerFunc **pfnBaseTransformer**
- void * **pBaseTransformerArg**
- int **bOwnSubtransformer**
- double **dfXOverviewFactor**
- double **dfYOverviewFactor**

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

- vrtwarped.cpp

Chapter 6

File Documentation

6.1 gdal_vrt.h File Reference

```
#include "gdal.h"
#include "cpl_port.h"
#include "cpl_error.h"
#include "cpl_minixml.h"
```

Macros

- #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

- CPL_C_START void **GDALRegister_VRT** (void)
- VRTDatasetH CPL_DLL CPL_STDCALL VRTCreate (int, int)
- void CPL_DLL CPL_STDCALL VRTFlushCache (VRTDatasetH)

- CPLXMLNode CPL_DLL *CPL_STDCALL VRTSerializeToXML (VRTDatasetH, const char *)
- int CPL_DLL CPL_STDCALL VRTAddBand (VRTDatasetH, GDALDataType, char **)
- CPLErr CPL_STDCALL VRTAddSource (VRTSourcedRasterBandH, VRTSourceH)
- CPLErr CPL_DLL CPL_STDCALL VRTAddSimpleSource (VRTSourcedRasterBandH, GDALRasterBandH, int, int, int, int, int, int, int, int, const char *, double)
- CPLErr CPL_DLL CPL_STDCALL VRTAddComplexSource (VRTSourcedRasterBandH, GDALRasterBandH, int, int, int, int, int, int, int, int, double, double, double)
- CPLErr CPL_DLL CPL_STDCALL VRTAddFuncSource (VRTSourcedRasterBandH, VRTImageReadFunc, void *, double)

6.1.1 Detailed Description

Public (C callable) entry points for virtual GDAL dataset objects.

6.1.2 Function Documentation

6.1.2.1 int CPL_DLL CPL_STDCALL VRTAddBand (VRTDatasetH *hDataset*, GDALDataType *eType*, char ** *papszOptions*)

See also

VRTDataset::VRTAddBand().

6.1.2.2 CPLErr CPL_DLL CPL_STDCALL 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().

6.1.2.3 CPLErr CPL_DLL CPL_STDCALL VRTAddFuncSource (VRTSourcedRasterBandH *hVRTBand*, VRTImageReadFunc *pfnReadFunc*, void * *pCBData*, double *dfNoDataValue*)

See also

VRTSourcedRasterBand::AddFuncSource().

6.1.2.4 CPLErr CPL_DLL CPL_STDCALL 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().

6.1.2.5 CPLErr CPL_STDCALL VRTAddSource (VRTSourcedRasterBandH *hVRTBand*, VRTSourceH *hNewSource*)

See also

VRTSourcedRasterBand::AddSource().

6.1.2.6 VRTDatasetH CPL_DLL CPL_STDCALL VRTCreate (int *nXSize,* int *nYSize*)

See also

VRTDataset::VRTDataset()

6.1.2.7 void CPL_DLL CPL_STDCALL VRTFlushCache (VRTDatasetH *hDataset*)

See also

VRTDataset::FlushCache()

6.1.2.8 CPLXMLNode CPL_DLL* CPL_STDCALL VRTSerializeToXML (VRTDatasetH *hDataset,* const char * *pszVRTPath*)

See also

VRTDataset::SerializeToXML()

Index