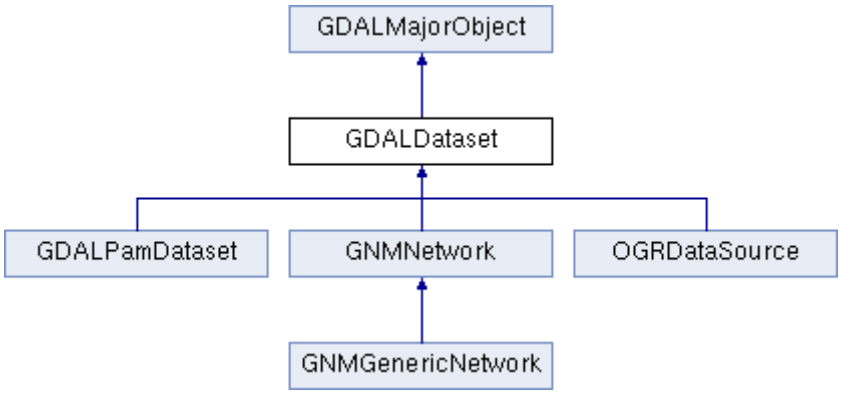
GDALDataset Class Reference

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

`#include <gdal_priv.h>`

Inheritance diagram for GDALDataset:



Classes

class	**Bands** Class returned by **GetBands()** that act as a container for raster bands. More...
struct	**FeatureLayerPair** Object returned by **GetFeatures()** iterators. More...
class	**Features** Class returned by **GetFeatures()** that act as a container for vector features. More...
class	**Layers** Class returned by **GetLayers()** that acts as a range of layers. More...

Public Member Functions

	~GDALDataset () override Destroy an open **GDALDataset**. More...
int	**GetRasterXSize** () Fetch raster width in pixels. More...
int	**GetRasterYSize** () Fetch raster height in pixels. More...
int	**GetRasterCount** () Fetch the number of raster bands on this dataset. More...
GDALRasterBand *	**GetRasterBand** (int) Fetch a band object for a dataset. More...
Bands	**GetBands** () Function that returns an iterable object over **GDALRasterBand** in the dataset. More...
virtual void	**FlushCache** (void) Flush all write cached data to disk. More...

virtual const char *	**GetProjectionRef** (void) Fetch the projection definition string for this dataset. More...
virtual **CPLErr**	**SetProjection** (const char *pszProjection) Set the projection reference string for this dataset. More...
virtual **CPLErr**	**GetGeoTransform** (double *padfTransform) Fetch the affine transformation coefficients. More...
virtual **CPLErr**	**SetGeoTransform** (double *padfTransform) Set the affine transformation coefficients. More...
virtual **CPLErr**	**AddBand** (**GDALDataType** eType, char **papszOptions=nullptr) Add a band to a dataset. More...
virtual void *	**GetInternalHandle** (const char *pszHandleName) Fetch a format specific internally meaningful handle. More...
virtual **GDALDriver** *	**GetDriver** (void) Fetch the driver to which this dataset relates. More...
virtual char **	**GetFileList** (void) Fetch files forming dataset. More...
virtual const char *	**GetDriverName** () Return driver name. More...
virtual int	**GetGCPCount** () Get number of GCPs. More...
virtual const char *	**GetGCPProjection** () Get output projection for GCPs. More...
virtual const **GDAL_GCP** *	**GetGCPs** () Fetch GCPs. More...
virtual **CPLErr**	**SetGCPs** (int nGCPCount, const **GDAL_GCP** *pasGCPList, const char *pszGCPProjection) Assign GCPs. More...
virtual **CPLErr**	**AdviseRead** (int nXOff, int nYOff, int nXSize, int nYSize, int nBufXSize, int nBufYSize, **GDALDataType** eDT, int nBandCount, int *panBandList, char **papszOptions) Advise driver of upcoming read requests. More...
virtual **CPLErr**	**CreateMaskBand** (int nFlagsIn) Adds a mask band to the dataset. More...
virtual **GDALAsyncReader** *	**BeginAsyncReader** (int nXOff, int nYOff, int nXSize, int nYSize, void *pBuf, int nBufXSize, int nBufYSize, **GDALDataType** eBufType, int nBandCount, int *panBandMap, int nPixelSpace, int nLineSpace, int nBandSpace, char **papszOptions) Sets up an asynchronous data request. More...
virtual void	**EndAsyncReader** (**GDALAsyncReader** *) End asynchronous request. More...
CPLErr	**RasterIO** (**GDALRWFlag**, int, int, int, int, void *, int, int, **GDALDataType**, int, int *, **GSpacing**, **GSpacing**, **GSpacing**, **GDALRasterIOExtraArg**

	*psExtraArg) **CPL_WARN_UNUSED_RESULT** Read/write a region of image data from multiple bands. More...
int	**Reference** () Add one to dataset reference count. More...
int	**Dereference** () Subtract one from dataset reference count. More...
int	**ReleaseRef** () Drop a reference to this object, and destroy if no longer referenced. More...
GDALAccess	**GetAccess** () const Return access mode. More...
int	**GetShared** () const Returns shared flag. More...
void	**MarkAsShared** () Mark this dataset as available for sharing.
void	**MarkSuppressOnClose** () Set that the dataset must be deleted on close. More...
char **	**GetOpenOptions** () Return open options. More...
CPLErr	**BuildOverviews** (const char *, int, int *, int, int *, GDALProgressFunc, void *) Build raster overview(s) More...
void	**ReportError** (**CPLErr** eErrClass, **CPLErrorNum** err_no, const char *fmt,...) **CPL_PRINT_FUNC_FORMAT**(4 Emits an error related to a dataset. More...
void char **	**GetMetadata** (const char *pszDomain="") override Fetch metadata. More...
CPLErr	**SetMetadata** (char **papszMetadata, const char *pszDomain) override Set metadata. More...
const char *	**GetMetadataItem** (const char *pszName, const char *pszDomain) override Fetch single metadata item. More...
CPLErr	**SetMetadataItem** (const char *pszName, const char *pszValue, const char *pszDomain) override Set single metadata item. More...
char **	**GetMetadataDomainList** () override Fetch list of metadata domains. More...
virtual int	**GetLayerCount** () Get the number of layers in this dataset. More...
virtual **OGRLayer** *	**GetLayer** (int iLayer) Fetch a layer by index. More...

Layers	**GetLayers** () Function that returns an iterable object over layers in the dataset. More...
virtual **OGRLayer** *	**GetLayerByName** (const char *) Fetch a layer by name. More...
virtual **OGRErr**	**DeleteLayer** (int iLayer) Delete the indicated layer from the datasource. More...
virtual void	**ResetReading** () Reset feature reading to start on the first feature. More...
virtual **OGRFeature** *	**GetNextFeature** (**OGRLayer** **ppoBelongingLayer, double *pdfProgressPct, GDALProgressFunc pfnProgress, void *pProgressData) Fetch the next available feature from this dataset. More...
Features	**GetFeatures** () Function that return an iterable object over features in the dataset layer. More...
virtual int	**TestCapability** (const char *) Test if capability is available. More...
virtual **OGRLayer** *	**CreateLayer** (const char *pszName, **OGRSpatialReference** *poSpatialRef=nullptr, **OGRwkbGeometryType** eGType=**wkbUnknown**, char **papszOptions=nullptr) This method attempts to create a new layer on the dataset with the indicated name, coordinate system, geometry type. More...
virtual **OGRLayer** *	**CopyLayer** (**OGRLayer** *poSrcLayer, const char *pszNewName, char **papszOptions=nullptr) Duplicate an existing layer. More...
virtual **OGRStyleTable** *	**GetStyleTable** () Returns dataset style table. More...
virtual void	**SetStyleTableDirectly** (**OGRStyleTable** *poStyleTable) Set dataset style table. More...
virtual void	**SetStyleTable** (**OGRStyleTable** *poStyleTable) Set dataset style table. More...
virtual **OGRLayer** *	**ExecuteSQL** (const char *pszStatement, **OGRGeometry** *poSpatialFilter, const char *pszDialect) Execute an SQL statement against the data store. More...
virtual void	**ReleaseResultSet** (**OGRLayer** *poResultsSet) Release results of **ExecuteSQL()**. More...
int	**GetRefCount** () const Fetch reference count. More...
int	**GetSummaryRefCount** () const Fetch reference count of datasource and all owned layers. More...
OGRErr	**Release** () Drop a reference to this dataset, and if the reference count drops to

	one close (destroy) the dataset. More...
virtual **OGRErr**	**StartTransaction** (int bForce=FALSE) For datasources which support transactions, StartTransaction creates a `transaction. More...
virtual **OGRErr**	**CommitTransaction** () For datasources which support transactions, CommitTransaction commits a transaction. More...
virtual **OGRErr**	**RollbackTransaction** () For datasources which support transactions, RollbackTransaction will roll back a datasource to its state before the start of the current transaction. More...

▸ **Public Member Functions inherited from GDALMajorObject**

Static Public Member Functions

static **GDALDataset** **	**GetOpenDatasets** (int *pnDatasetCount) Fetch all open GDAL dataset handles. More...
static **GDALDatasetH**	**ToHandle** (**GDALDataset** *poDS) Convert a GDALDataset* to a GDALDatasetH. More...
static **GDALDataset** *	**FromHandle** (**GDALDatasetH** hDS) Convert a GDALDatasetH to a GDALDataset*. More...
static **GDALDataset** *	**Open** (const char *pszFilename, unsigned int nOpenFlags=0, const char *const *papszAllowedDrivers=nullptr, const char *const *papszOpenOptions=nullptr, const char *const *papszSiblingFiles=nullptr)

▸ **Static Public Member Functions inherited from GDALMajorObject**

Protected Member Functions

virtual int	**CloseDependentDatasets** () Drop references to any other datasets referenced by this dataset. More...
virtual **OGRLayer** *	**ICreateLayer** (const char *pszName, **OGRSpatialReference** *poSpatialRef=nullptr, **OGRwkbGeometryType** eGType=**wkbUnknown**, char **papszOptions=nullptr) This method attempts to create a new layer on the dataset with the indicated name, coordinate system, geometry type. More...

▸ **Protected Member Functions inherited from GDALMajorObject**

Friends

class	**GDALDriver**
class	**GDALDefaultOverviews**
class	**GDALProxyDataset**
class	**GDALDriverManager**
GDALDatasetH	**GDALOpenEx** (const char *pszFilename, unsigned int nOpenFlags, const char *const

	*papszAllowedDrivers, const char *const *papszOpenOptions, const char *const *papszSiblingFiles) Open a raster or vector file as a **GDALDataset**. More...
void	**GDALClose** (**GDALDatasetH** hDS) Close GDAL dataset. More...

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

Constructor & Destructor Documentation

GDALDataset::~GDALDataset ( ) override

Destroy an open **GDALDataset**.

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

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

For Windows users, it is not recommended to use the delete operator on the dataset object because of known issues when allocating and freeing memory across module boundaries. Calling **GDALClose()** is then a better option.

Member Function Documentation

```
CPLErr GDALDataset::AddBand ( GDALDataType  eType,
                              char **       papszOptions = nullptr
                            )                                          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** is not returned. It may be fetched after successful completion of the method by calling GDALDataset::GetRasterBand(GDALDataset::GetRasterCount()) 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.

```
CPLErr GDALDataset::AdviseRead ( int           nXOff,
                                 int           nYOff,
                                 int           nXSize,
                                 int           nYSize,
                                 int           nBufXSize,
                                 int           nBufYSize,
                                 GDALDataType  eBufType,
                                 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()** 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()** call, but it can dramatically accelerate access via some drivers.

Depending on call paths, drivers might receive several calls to **AdviseRead()** with the same parameters.

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** 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.

```
GDALAsyncReader * GDALDataset::BeginAsyncReader ( int          nXOff,
                                                  int          nYOff,
                                                  int          nXSize,
                                                  int          nYSize,
                                                  void *       pBuf,
                                                  int          nBufXSize,
                                                  int          nBufYSize,
                                                  GDALDataType eBufType,
                                                  int          nBandCount,
                                                  int *        panBandMap,
                                                  int          nPixelSpace,
                                                  int          nLineSpace,
                                                  int          nBandSpace,
                                                  char **      papszOptions
                                                )                                virtual
```

Sets up an asynchronous data request.

This method establish an asynchronous raster read request for the indicated window on the dataset into the indicated buffer. The parameters for windowing, buffer size, buffer type and buffer organization are similar to those for **GDALDataset::RasterIO()**; however, this call only launches the request and filling the buffer is accomplished via calls to GetNextUpdatedRegion() on the return **GDALAsyncReader** session object.

Once all processing for the created session is complete, or if no further refinement of the request is required, the **GDALAsyncReader** object should be destroyed with the **GDALDataset::EndAsyncReader()** method.

Note that the data buffer (pData) will potentially continue to be updated as long as the session lives, but it is not deallocated when the session (**GDALAsyncReader**) is destroyed with **EndAsyncReader()**. It should be deallocated by the application at that point.

Additional information on asynchronous IO in GDAL may be found at:
http://trac.osgeo.org/gdal/wiki/rfc24_progressive_data_support

This method is the same as the C **GDALBeginAsyncReader()** function.

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.
pBuf	The buffer into which the data should be read. 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** 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.
papszOptions	Driver specific control options in a string list or NULL. Consult driver documentation for options supported.

Returns

The **GDALAsyncReader** object representing the request.

```
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()** will return CPLE_NotSupported.

Depending on the actual file format, all overviews level can be also deleted by specifying nOverviews == 0. This works at least for external overviews (.ovr), TIFF internal overviews, etc.

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

Parameters

pszResampling	one of "AVERAGE", "AVERAGE_MAGPHASE", "BILINEAR", "CUBIC", "CUBICSPLINE", "GAUSS", "LANCZOS", "MODE", "NEAREST", or "NONE" controlling the downsampling method applied.
nOverviews	number of overviews to build, or 0 to clean overviews.
panOverviewList	the list of overview decimation factors to build, or NULL if nOverviews == 0.
nListBands	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, nullptr,
                           GDALDummyProgress, nullptr );
```

See also

GDALRegenerateOverviews()

int GDALDataset::CloseDependentDatasets ( ) protected virtual

Drop references to any other datasets referenced by this dataset.

This method should release any reference to other datasets (e.g. a VRT dataset to its sources), but not close the current dataset itself.

If at least, one reference to a dependent dataset has been dropped, this method should return TRUE. Otherwise it *should* return FALSE. (Failure to return the proper value might result in infinite loop)

This method can be called several times on a given dataset. After the first time, it should not do anything and return FALSE.

The driver implementation may choose to destroy its raster bands, so be careful not to call any method on the raster bands afterwards.

Basically the only safe action you can do after calling **CloseDependentDatasets()** is to call the destructor.

Note: the only legitimate caller of **CloseDependentDatasets()** is GDALDriverManager::~GDALDriverManager()

Returns

TRUE if at least one reference to another dataset has been dropped.

Reimplemented in **GNMGenericNetwork**.

OGRErr GDALDataset::CommitTransaction ( ) virtual

For datasources which support transactions, CommitTransaction commits a transaction.

If no transaction is active, or the commit fails, will return OGRERR_FAILURE. Datasources which do not support transactions will always return OGRERR_UNSUPPORTED_OPERATION.

Depending on drivers, this may or may not abort layer sequential readings that are active.

This function is the same as the C function **GDALDatasetCommitTransaction()**.

Returns

OGRERR_NONE on success.

Since

GDAL 2.0

```
OGRLayer * GDALDataset::CopyLayer ( OGRLayer *  poSrcLayer,
                                    const char *  pszNewName,
                                    char **  papszOptions = nullptr
                                  )                                  virtual
```

Duplicate an existing layer.

This method creates a new layer, duplicate the field definitions of the source layer and then duplicate each features of the source layer. The papszOptions argument can be used to control driver specific creation options. These options are normally documented in the format specific documentation. The source layer may come from another dataset.

This method is the same as the C function **GDALDatasetCopyLayer()** and the deprecated **OGR_DS_CopyLayer()**.

In GDAL 1.X, this method used to be in the **OGRDataSource** class.

Parameters

poSrcLayer	source layer.
pszNewName	the name of the layer to create.
papszOptions	a StringList of name=value options. Options are driver specific. There is a common option to set output layer spatial reference: DST_SRSWKT. The option should be in WKT format.

Returns

an handle to the layer, or NULL if an error occurs.

Reimplemented in **GNMGenericNetwork**.

```
OGRLayer *
GDALDataset::CreateLayer          ( const char *            pszName,
                                    OGRSpatialReference *   poSpatialRef = nullptr,
                                    OGRwkbGeometryType      eGType = wkbUnknown,
                                    char **                 papszOptions = nullptr
                                  )                                                    virtual
```

This method attempts to create a new layer on the dataset with the indicated name, coordinate system, geometry type.

The papszOptions argument can be used to control driver specific creation options. These options are normally documented in the format specific documentation.

In GDAL 2.0, drivers should extend the **ICreateLayer()** method and not **CreateLayer()**. **CreateLayer()** adds validation of layer creation options, before delegating the actual work to **ICreateLayer()**.

This method is the same as the C function **GDALDatasetCreateLayer()** and the deprecated **OGR_DS_CreateLayer()**.

In GDAL 1.X, this method used to be in the **OGRDataSource** class.

Parameters

pszName	the name for the new layer. This should ideally not match any existing layer on the datasource.
poSpatialRef	the coordinate system to use for the new layer, or NULL if no coordinate system is available. The driver might only increase the reference counter of the object to take ownership, and not make a full copy, so do not use **OSRDestroySpatialReference()**, but **OSRRelease()** instead when you are done with the object.
eGType	the geometry type for the layer. Use wkbUnknown if there are no constraints on the types geometry to be written.
papszOptions	a StringList of name=value options. Options are driver specific.

Returns

NULL is returned on failure, or a new **OGRLayer** handle on success.

Example:

```
#include "gdal.h"
#include "cpl_string.h"

...

        OGRLayer *poLayer;
        char     **papszOptions;

        if( !poDS->TestCapability( ODsCCreateLayer ) )
        {
        ...
        }

        papszOptions = CSLSetNameValue( papszOptions, "DIM", "2" );
        poLayer = poDS->CreateLayer( "NewLayer", nullptr, wkbUnknown,
                                     papszOptions );
```

```
CSLDestroy( papszOptions );

if( poLayer == NULL )
{
    ...
}
```

CPLErr GDALDataset::CreateMaskBand (int nFlagsIn) virtual

Adds a mask band to the dataset.

The default implementation of the **CreateMaskBand()** method is implemented based on similar rules to the .ovr handling implemented using the GDALDefaultOverviews object. A TIFF file with the extension .msk will be created with the same basename as the original file, and it will have one band. The mask images will be deflate compressed tiled images with the same block size as the original image if possible. It will have INTERNAL_MASK_FLAGS_xx metadata items set at the dataset level, where xx matches the band number of a band of the main dataset. The value of those items will be the one of the nFlagsIn parameter.

Note that if you got a mask band with a previous call to GetMaskBand(), it might be invalidated by **CreateMaskBand()**. So you have to call GetMaskBand() again.

Since

GDAL 1.5.0

Parameters

nFlagsIn 0 or combination of GMF_PER_DATASET / GMF_ALPHA. GMF_PER_DATASET will be always set, even if not explicitly specified.

Returns

CE_None on success or CE_Failure on an error.

See also

http://trac.osgeo.org/gdal/wiki/rfc15_nodatabitmask

GDALRasterBand::CreateMaskBand()

OGRErr GDALDataset::DeleteLayer (int iLayer) virtual

Delete the indicated layer from the datasource.

If this method is supported the ODsCDeleteLayer capability will test TRUE on the **GDALDataset**.

This method is the same as the C function **GDALDatasetDeleteLayer()** and the deprecated **OGR_DS_DeleteLayer()**.

In GDAL 1.X, this method used to be in the **OGRDataSource** class.

Parameters

iLayer the index of the layer to delete.

Returns

OGRERR_NONE on success, or OGRERR_UNSUPPORTED_OPERATION if deleting layers is not supported for this datasource.

Reimplemented in **GNMGenericNetwork**.

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()** function.

Returns

the post-decrement reference count.

void GDALDataset::EndAsyncReader (GDALAsyncReader * poARIO) virtual

End asynchronous request.

This method destroys an asynchronous io request and recovers all resources associated with it.

This method is the same as the C function **GDALEndAsyncReader()**.

Parameters

poARIO pointer to a **GDALAsyncReader**

```
OGRLayer * GDALDataset::ExecuteSQL ( const char *      pszStatement,
                                     OGRGeometry *    poSpatialFilter,
                                     const char *     pszDialect
                                   )                                    virtual
```

Execute an SQL statement against the data store.

The result of an SQL query is either NULL for statements that are in error, or that have no results set, or an **OGRLayer** pointer representing a results set from the query. Note that this **OGRLayer** is in addition to the layers in the data store and must be destroyed with **ReleaseResultSet()** before the dataset is closed (destroyed).

This method is the same as the C function **GDALDatasetExecuteSQL()** and the deprecated **OGR_DS_ExecuteSQL()**.

For more information on the SQL dialect supported internally by OGR review the OGR SQL document. Some drivers (i.e. Oracle and PostGIS) pass the SQL directly through to the underlying RDBMS.

Starting with OGR 1.10, the SQLITE dialect can also be used.

In GDAL 1.X, this method used to be in the **OGRDataSource** class.

Parameters

pszStatement	the SQL statement to execute.
poSpatialFilter	geometry which represents a spatial filter. Can be NULL.
pszDialect	allows control of the statement dialect. If set to NULL, the OGR SQL engine will be used, except for RDBMS drivers that will use their dedicated SQL engine, unless OGRSQL is explicitly passed as the dialect. Starting with OGR 1.10, the SQLITE dialect can also be used.

Returns

an **OGRLayer** containing the results of the query. Deallocate with **ReleaseResultSet()**.

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.

The default implementation of this method just calls the **FlushCache()** method on each of the raster bands and the SyncToDisk() method on each of the layers. Conceptionally, calling **FlushCache()** on a dataset should include any work that might be accomplished by calling SyncToDisk() on layers in that dataset.

Using this method does not prevent use from calling **GDALClose()** to properly close a dataset and ensure that important data not addressed by **FlushCache()** is written in the file.

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

Reimplemented in **GNMGenericNetwork**, and **GDALPamDataset**.

static GDALDataset* GDALDataset::FromHandle (GDALDatasetH hDS) inline static

Convert a GDALDatasetH to a GDALDataset*.

Since
GDAL 2.3

GDALAccess GDALDataset::GetAccess ( ) const inline

Return access mode.

Returns
access mode.

GDALDataset::Bands GDALDataset::GetBands ( )

Function that returns an iterable object over **GDALRasterBand** in the dataset.

This is a C++ iterator friendly version of **GetRasterBand()**.

Typical use is:

```
for( auto&& poBand: poDS->GetBands() )
{
      std::cout << "Band  << poBand->GetDescription() << std::endl;
}
```

See also

GetRasterBand()

Since

GDAL 2.3

GDALDriver * GDALDataset::GetDriver (void) virtual

Fetch the driver to which this dataset relates.

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

Returns

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

const char * GDALDataset::GetDriverName ( ) virtual

Return driver name.

Returns

driver name.

GDALDataset::Features GDALDataset::GetFeatures ( )

Function that return an iterable object over features in the dataset layer.

This is a C++ iterator friendly version of **GetNextFeature()**.

Using this iterator for standard range-based loops is safe, but due to implementation limitations, you shouldn't try to access (dereference) more than one iterator step at a time, since the **FeatureLayerPair** reference which is returned is reused.

Typical use is:

```
for( auto&& oFeatureLayerPair: poDS->GetFeatures() )
{
      std::cout << "Feature of layer " <<
              oFeatureLayerPair.layer->GetName() << std::endl;
      oFeatureLayerPair.feature->DumpReadable();
}
```

See also

GetNextFeature()

Since

GDAL 2.3

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()**.

The returned filenames will normally be relative or absolute paths depending on the path used to originally open the dataset. The strings will be UTF-8 encoded.

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

Returns

NULL or a NULL terminated array of file names.

Reimplemented in **GDALPamDataset**, and **GNMNetwork**.

int GDALDataset::GetGCPCount ( ) virtual

Get number of GCPs.

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

Returns

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

Reimplemented in **GDALPamDataset**.

const char * GDALDataset::GetGCPProjection ( ) virtual

Get output projection for GCPs.

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

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

Returns

internal projection string or "" if there are no GCPs. It should not be altered, freed or expected to last for long.

Reimplemented in **GDALPamDataset**.

const GDAL_GCP * GDALDataset::GetGCPs ( ) virtual

Fetch GCPs.

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

Returns

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

Reimplemented in **GDALPamDataset**.

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.

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

void * GDALDataset::GetInternalHandle (const char * pszHandleName) virtual

Fetch a format specific internally meaningful handle.

This method is the same as the C **GDALGetInternalHandle()** 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 recognized/supported.

OGRLayer * GDALDataset::GetLayer (int iLayer) virtual

Fetch a layer by index.

The returned layer remains owned by the GDALDataset and should not be deleted by the application.

See GetLayers() for a C++ iterator version of this method.

This method is the same as the C function GDALDatasetGetLayer() and the deprecated OGR_DS_GetLayer().

In GDAL 1.X, this method used to be in the OGRDataSource class.

Parameters

iLayer a layer number between 0 and GetLayerCount()-1.

Returns

the layer, or NULL if iLayer is out of range or an error occurs.

See also

GetLayers()

Reimplemented in GNMGenericNetwork.

OGRLayer * GDALDataset::GetLayerByName (const char * pszName) virtual

Fetch a layer by name.

The returned layer remains owned by the GDALDataset and should not be deleted by the application.

This method is the same as the C function GDALDatasetGetLayerByName() and the deprecated OGR_DS_GetLayerByName().

In GDAL 1.X, this method used to be in the OGRDataSource class.

Parameters

pszName the layer name of the layer to fetch.

Returns

the layer, or NULL if Layer is not found or an error occurs.

int GDALDataset::GetLayerCount ( ) virtual

Get the number of layers in this dataset.

This method is the same as the C function **GDALDatasetGetLayerCount()**, and the deprecated **OGR_DS_GetLayerCount()**.

In GDAL 1.X, this method used to be in the **OGRDataSource** class.

Returns

layer count.

Reimplemented in **GNMGenericNetwork**.

GDALDataset::Layers GDALDataset::GetLayers ( )

Function that returns an iterable object over layers in the dataset.

This is a C++ iterator friendly version of **GetLayer()**.

Typical use is:

```
for( auto&& poLayer: poDS->GetLayers() )
{
      std::cout << "Layer  << poLayer->GetName() << std::endl;
}
```

See also

GetLayer()

Since

GDAL 2.3

char ** GDALDataset::GetMetadata (const char * pszDomain = "") override virtual

Fetch metadata.

The returned string list is owned by the object, and may change at any time. It is formatted as a "Name=value" list with the last pointer value being NULL. Use 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()**.

Parameters

pszDomain the domain of interest. Use "" or NULL for the default domain.

Returns

NULL or a string list.

Reimplemented from **GDALMajorObject**.

char ** GDALDataset::GetMetadataDomainList ( ) override virtual

Fetch list of metadata domains.

The returned string list is the list of (non-empty) metadata domains.

This method does the same thing as the C function **GDALGetMetadataDomainList()**.

Returns

NULL or a string list. Must be freed with **CSLDestroy()**

Since

GDAL 1.11

Reimplemented from **GDALMajorObject**.

const char* GDALDataset::GetMetadataItem (const char * pszName,
const char * pszDomain
) override virtual

Fetch single metadata item.

The C function **GDALGetMetadataItem()** 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.

Reimplemented from **GDALMajorObject**.

```
OGRFeature * GDALDataset::GetNextFeature ( OGRLayer **         ppoBelongingLayer,
                                           double *            pdfProgressPct,
                                           GDALProgressFunc    pfnProgress,
                                           void *              pProgressData
                                         )                                          virtual
```

Fetch the next available feature from this dataset.

This method is intended for the few drivers where **OGRLayer::GetNextFeature()** is not efficient, but in general **OGRLayer::GetNextFeature()** is a more natural API.

See **GetFeatures()** for a C++ iterator version of this method.

The returned feature becomes the responsibility of the caller to delete with **OGRFeature::DestroyFeature()**.

Depending on the driver, this method may return features from layers in a non sequential way. This is what may happen when the ODsCRandomLayerRead capability is declared (for example for the OSM and GMLAS drivers). When datasets declare this capability, it is strongly advised to use **GDALDataset::GetNextFeature()** instead of **OGRLayer::GetNextFeature()**, as the later might have a slow, incomplete or stub implementation.

The default implementation, used by most drivers, will however iterate over each layer, and then over each feature within this layer.

This method takes into account spatial and attribute filters set on layers that will be iterated upon.

The **ResetReading()** method can be used to start at the beginning again.

Depending on drivers, this may also have the side effect of calling **OGRLayer::GetNextFeature()** on the layers of this dataset.

This method is the same as the C function **GDALDatasetGetNextFeature()**.

Parameters

ppoBelongingLayer	a pointer to a OGRLayer* variable to receive the layer to which the object belongs to, or NULL. It is possible that the output of *ppoBelongingLayer to be NULL despite the feature not being NULL.
pdfProgressPct	a pointer to a double variable to receive the percentage progress (in [0,1] range), or NULL. On return, the pointed value might be negative if determining the progress is not possible.
pfnProgress	a progress callback to report progress (for **GetNextFeature()** calls that might have a long duration) and offer cancellation possibility, or NULL.
pProgressData	user data provided to pfnProgress, or NULL

Returns

a feature, or NULL if no more features are available.

Since

GDAL 2.2

See also

GetFeatures()

GDALDataset ** GDALDataset::GetOpenDatasets (int * pnCount) static

Fetch all open GDAL dataset handles.

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

NOTE: This method is not thread safe. The returned list may change at any time and it should not be freed.

Parameters

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

Returns

a pointer to an array of dataset handles.

char** GDALDataset::GetOpenOptions ( ) inline

Return open options.

Returns

open options.

const char * GDALDataset::GetProjectionRef (void) virtual

Fetch the projection definition string for this dataset.

Same as the C function GDALGetProjectionRef().

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/osr_tutorial.html

Reimplemented in GDALPamDataset, and GNMNetwork.

GDALRasterBand * GDALDataset::GetRasterBand (int nBandId)

Fetch a band object for a dataset.

See GetBands() for a C++ iterator version of this method.

Equivalent of the C function GDALGetRasterBand().

Parameters

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

Returns

the nBandId th band object

int GDALDataset::GetRasterCount ( )

Fetch the number of raster bands on this dataset.

Same as the C function GDALGetRasterCount().

Returns

the number of raster bands.

int GDALDataset::GetRasterXSize ( )

Fetch raster width in pixels.

Equivalent of the C function GDALGetRasterXSize().

Returns

the width in pixels of raster bands in this GDALDataset.

int GDALDataset::GetRasterYSize ( )

Fetch raster height in pixels.

Equivalent of the C function GDALGetRasterYSize().

Returns

the height in pixels of raster bands in this GDALDataset.

int GDALDataset::GetRefCount ( ) const

Fetch reference count.

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

In GDAL 1.X, this method used to be in the **OGRDataSource** class.

Returns

the current reference count for the datasource object itself.

int GDALDataset::GetShared ( ) const

Returns shared flag.

Returns

TRUE if the **GDALDataset** is available for sharing, or FALSE if not.

OGRStyleTable * GDALDataset::GetStyleTable ( ) virtual

Returns dataset style table.

This method is the same as the C function **GDALDatasetGetStyleTable()** and the deprecated **OGR_DS_GetStyleTable()**.

In GDAL 1.X, this method used to be in the **OGRDataSource** class.

Returns

pointer to a style table which should not be modified or freed by the caller.

int GDALDataset::GetSummaryRefCount ( ) const

Fetch reference count of datasource and all owned layers.

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

In GDAL 1.X, this method used to be in the **OGRDataSource** class.

Deprecated:

Returns

the current summary reference count for the datasource and its layers.

```
OGRLayer *
GDALDataset::ICreateLayer ( const char *          pszName,
                            OGRSpatialReference * poSpatialRef = nullptr,
                            OGRwkbGeometryType    eGType = wkbUnknown,
                            char **               papszOptions = nullptr
                          )
```

protected virtual

This method attempts to create a new layer on the dataset with the indicated name, coordinate system, geometry type.

This method is reserved to implementation by drivers.

The papszOptions argument can be used to control driver specific creation options. These options are normally documented in the format specific documentation.

Parameters

pszName	the name for the new layer. This should ideally not match any existing layer on the datasource.
poSpatialRef	the coordinate system to use for the new layer, or NULL if no coordinate system is available.
eGType	the geometry type for the layer. Use wkbUnknown if there are no constraints on the types geometry to be written.
papszOptions	a StringList of name=value options. Options are driver specific.

Returns

NULL is returned on failure, or a new **OGRLayer** handle on success.

Since

GDAL 2.0

```
void GDALDataset::MarkSuppressOnClose ( )
```

inline

Set that the dataset must be deleted on close.

```
static GDALDataset*
GDALDataset::Open               ( const char *          pszFilename,
                                  unsigned int          nOpenFlags = 0,
                                  const char *const *   papszAllowedDrivers = nullptr,
                                  const char *const *   papszOpenOptions = nullptr,
                                  const char *const *   papszSiblingFiles = nullptr
                                )                                                    inline static
```

See also

GDALOpenEx().

Since

GDAL 2.3

```
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,
                               GSpacing            nPixelSpace,
                               GSpacing            nLineSpace,
                               GSpacing            nBandSpace,
                               GDALRasterIOExtraArg * psExtraArg
                             )
```

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**. 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()** or **GDALDatasetRasterIOEx()** functions.

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** 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 (0) 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 (0) the value will be nLineSpace * nBufYSize implying band sequential organization of the data buffer.

psExtraArg (new in GDAL 2.0) pointer to a **GDALRasterIOExtraArg** structure with additional arguments to specify resampling and progress callback, or NULL for default behaviour. The GDAL_RASTERIO_RESAMPLING configuration option can also be defined to override the default resampling to one of BILINEAR, CUBIC, CUBICSPLINE, LANCZOS, AVERAGE or MODE.

Returns

CE_Failure if the access fails, otherwise CE_None.

int GDALDataset::Reference ( )

Add one to dataset reference count.

The reference is one after instantiation.

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

Returns

the post-increment reference count.

OGRErr GDALDataset::Release ( )

Drop a reference to this dataset, and if the reference count drops to one close (destroy) the dataset.

This method is the same as the C function **OGRReleaseDataSource()**.

Deprecated:
. In GDAL 2, use **GDALClose()** instead

Returns
OGRERR_NONE on success or an error code.

int GDALDataset::ReleaseRef ( )

Drop a reference to this object, and destroy if no longer referenced.

Returns
TRUE if the object has been destroyed.

Since
GDAL 2.2

void GDALDataset::ReleaseResultSet (OGRLayer * poResultsSet) virtual

Release results of **ExecuteSQL()**.

This method should only be used to deallocate OGRLayers resulting from an **ExecuteSQL()** call on the same **GDALDataset**. Failure to deallocate a results set before destroying the **GDALDataset** may cause errors.

This method is the same as the C function **GDALDatasetReleaseResultSet()** and the deprecated **OGR_DS_ReleaseResultSet()**.

In GDAL 1.X, this method used to be in the **OGRDataSource** class.

Parameters
poResultsSet the result of a previous **ExecuteSQL()** call.

void GDALDataset::ReportError (CPLErr eErrClass, CPLErrorNum err_no, const char * fmt, ...)

Emits an error related to a dataset.

This function is a wrapper for regular **CPLError()**. The only difference with **CPLError()** is that it prepends the error message with the dataset name.

Parameters

eErrClass	one of CE_Warning, CE_Failure or CE_Fatal.
err_no	the error number (CPLE_*) from **cpl_error.h**.
fmt	a printf() style format string. Any additional arguments will be treated as arguments to fill in this format in a manner similar to printf().

Since

GDAL 1.9.0

void GDALDataset::ResetReading ( ) virtual

Reset feature reading to start on the first feature.

This affects **GetNextFeature()**.

Depending on drivers, this may also have the side effect of calling **OGRLayer::ResetReading()** on the layers of this dataset.

This method is the same as the C function **GDALDatasetResetReading()**.

Since

GDAL 2.2

OGRErr GDALDataset::RollbackTransaction ( ) virtual

For datasources which support transactions, RollbackTransaction will roll back a datasource to its state before the start of the current transaction.

If no transaction is active, or the rollback fails, will return OGRERR_FAILURE. Datasources which do not support transactions will always return OGRERR_UNSUPPORTED_OPERATION.

This function is the same as the C function **GDALDatasetRollbackTransaction()**.

Returns

OGRERR_NONE on success.

Since

GDAL 2.0

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()**.

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 responsible for deallocating these arguments if appropriate.

Most formats do not support setting of GCPs, even formats 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**.

CPLErr GDALDataset::SetGeoTransform (double * padfTransform) virtual

Set the affine transformation coefficients.

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

This method does the same thing as the C GDALSetGeoTransform() 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.

GDALDataset::SetMetadata (char ** papszMetadata, const char * pszDomain) override virtual

Set metadata.

CAUTION: depending on the format, older values of the updated information might still be found in the file in a "ghost" state, even if no longer accessible through the GDAL API. This is for example the case of the GTiff format (this is not a exhaustive list)

The C function GDALSetMetadata() 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.

GDALDataset::SetMetadataItem (const char * pszName,
const char * pszValue,
const char * pszDomain
) override virtual

Set single metadata item.

CAUTION: depending on the format, older values of the updated information might still be found in the file in a "ghost" state, even if no longer accessible through the GDAL API. This is for example the case of the GTiff format (this is not a exhaustive list)

The C function **GDALSetMetadataItem()** 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**.

CPLErr GDALDataset::SetProjection (const char * pszProjection) 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()** function.

Parameters

pszProjection projection reference string.

Returns

CE_Failure if an error occurs, otherwise CE_None.

Reimplemented in **GDALPamDataset**.

void GDALDataset::SetStyleTable (OGRStyleTable * poStyleTable) virtual

Set dataset style table.

This method operate exactly as **SetStyleTableDirectly()** except that it does not assume ownership of the passed table.

This method is the same as the C function **GDALDatasetSetStyleTable()** and the deprecated **OGR_DS_SetStyleTable()**.

In GDAL 1.X, this method used to be in the **OGRDataSource** class.

Parameters

poStyleTable pointer to style table to set

void GDALDataset::SetStyleTableDirectly (OGRStyleTable * poStyleTable) virtual

Set dataset style table.

This method operate exactly as **SetStyleTable()** except that it assumes ownership of the passed table.

This method is the same as the C function **GDALDatasetSetStyleTableDirectly()** and the deprecated **OGR_DS_SetStyleTableDirectly()**.

In GDAL 1.X, this method used to be in the **OGRDataSource** class.

Parameters

poStyleTable pointer to style table to set

OGRErr GDALDataset::StartTransaction (int bForce = FALSE) virtual

For datasources which support transactions, StartTransaction creates a `transaction.

If starting the transaction fails, will return OGRERR_FAILURE. Datasources which do not support transactions will always return OGRERR_UNSUPPORTED_OPERATION.

Nested transactions are not supported.

All changes done after the start of the transaction are definitely applied in the datasource if **CommitTransaction()** is called. They may be canceled by calling **RollbackTransaction()** instead.

At the time of writing, transactions only apply on vector layers.

Datasets that support transactions will advertise the ODsCTransactions capability. Use of transactions at dataset level is generally preferred to transactions at layer level, whose scope is rarely limited to the layer from which it was started.

In case **StartTransaction()** fails, neither **CommitTransaction()** or **RollbackTransaction()** should be called.

If an error occurs after a successful **StartTransaction()**, the whole transaction may or may not be implicitly canceled, depending on drivers. (e.g. the PG driver will cancel it, SQLite/GPKG not). In any case, in the event of an error, an explicit call to **RollbackTransaction()** should be done to keep things balanced.

By default, when bForce is set to FALSE, only "efficient" transactions will be attempted. Some drivers may offer an emulation of transactions, but sometimes with significant overhead, in which case the user must explicitly allow for such an emulation by setting bForce to TRUE. Drivers that offer emulated transactions should advertise the ODsCEmulatedTransactions capability (and not ODsCTransactions).

This function is the same as the C function **GDALDatasetStartTransaction()**.

Parameters

bForce can be set to TRUE if an emulation, possibly slow, of a transaction mechanism is acceptable.

Returns

OGRERR_NONE on success.

Since

GDAL 2.0

int GDALDataset::TestCapability (const char * pszCap) virtual

Test if capability is available.

One of the following dataset capability names can be passed into this method, and a TRUE or FALSE value will be returned indicating whether or not the capability is available for this object.

- **ODsCCreateLayer**: True if this datasource can create new layers.
- **ODsCDeleteLayer**: True if this datasource can delete existing layers.
- **ODsCCreateGeomFieldAfterCreateLayer**: True if the layers of this datasource support CreateGeomField() just after layer creation.
- **ODsCCurveGeometries**: True if this datasource supports curve geometries.
- **ODsCTransactions**: True if this datasource supports (efficient) transactions.
- **ODsCEmulatedTransactions**: True if this datasource supports transactions through emulation.
- **ODsCRandomLayerRead**: True if this datasource has a dedicated **GetNextFeature()** implementation, potentially returning features from layers in a non sequential way.
- **ODsCRandomLayerWrite**: True if this datasource supports calling CreateFeature() on layers in a non sequential way.

The #define macro forms of the capability names should be used in preference to the strings themselves to avoid misspelling.

This method is the same as the C function **GDALDatasetTestCapability()** and the deprecated **OGR_DS_TestCapability()**.

In GDAL 1.X, this method used to be in the **OGRDataSource** class.

Parameters

pszCap the capability to test.

Returns

TRUE if capability available otherwise FALSE.

Reimplemented in **GNMGenericNetwork**.

static GDALDatasetH GDALDataset::ToHandle (GDALDataset * poDS) inline static

Convert a GDALDataset* to a GDALDatasetH.

Since

GDAL 2.3

Friends And Related Function Documentation

void GDALClose (GDALDatasetH hDS) friend

Close GDAL dataset.

For non-shared datasets (opened with **GDALOpen()**) the dataset is closed using the C++ "delete" operator, recovering all dataset related resources. For shared datasets (opened with **GDALOpenShared()**) 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 *".

```
GDALDatasetH GDALOpenEx ( const char *          pszFilename,
                          unsigned int          nOpenFlags,
                          const char *const *   papszAllowedDrivers,
                          const char *const *   papszOpenOptions,
                          const char *const *   papszSiblingFiles
                        )                                                  friend
```

Open a raster or vector file as a **GDALDataset**.

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

Several recommendations :

- If you open a dataset object with GDAL_OF_UPDATE access, it is not recommended 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.)

For drivers supporting the VSI virtual file API, it is possible to open a file in a .zip archive (see **VSIInstallZipFileHandler()**), in a .tar/.tar.gz/.tgz archive (see **VSIInstallTarFileHandler()**) or on a HTTP / FTP server (see **VSIInstallCurlFileHandler()**)

In some situations (dealing with unverified data), the datasets can be opened in another process through the **GDAL API Proxy** mechanism.

In order to reduce the need for searches through the operating system file system machinery, it is possible to give an optional list of files with the papszSiblingFiles parameter. 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 must not include any path components, are essentially just the output of **VSIReadDir()** 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. It should be in UTF-8 encoding.
nOpenFlags	a combination of GDAL_OF_ flags that may be combined through logical or operator. • Driver kind: GDAL_OF_RASTER for raster drivers, GDAL_OF_VECTOR for vector drivers, GDAL_OF_GNM for Geographic Network Model drivers. If none of the value is specified, all kinds are implied. • Access mode: GDAL_OF_READONLY (exclusive)or GDAL_OF_UPDATE.

- Shared mode: GDAL_OF_SHARED. If set, it allows the sharing of **GDALDataset** handles for a dataset with other callers that have set GDAL_OF_SHARED. In particular, **GDALOpenEx()** will first consult its list of currently open and shared **GDALDataset**'s, and if the **GetDescription()** name for one exactly matches the pszFilename passed to **GDALOpenEx()** it will be referenced and returned, if **GDALOpenEx()** is called from the same thread.
- Verbose error: GDAL_OF_VERBOSE_ERROR. If set, a failed attempt to open the file will lead to an error message to be reported.

papszAllowedDrivers	NULL to consider all candidate drivers, or a NULL terminated list of strings with the driver short names that must be considered.
papszOpenOptions	NULL, or a NULL terminated list of strings with open options passed to candidate drivers. An option exists for all drivers, OVERVIEW_LEVEL=level, to select a particular overview level of a dataset. The level index starts at 0. The level number can be suffixed by "only" to specify that only this overview level must be visible, and not sub-levels. Open options are validated by default, and a warning is emitted in case the option is not recognized. In some scenarios, it might be not desirable (e.g. when not knowing which driver will open the file), so the special open option VALIDATE_OPEN_OPTIONS can be set to NO to avoid such warnings. Alternatively, since GDAL 2.1, an option name can be preceded by the @ character to indicate that it may not cause a warning if the driver doesn't declare this option.
papszSiblingFiles	NULL, or a NULL terminated list of strings that are filenames that are auxiliary to the main filename. If NULL is passed, a probing of the file system will be done.

Returns

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

Since

GDAL 2.0

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

- **gdal_priv.h**
- gdaldataset.cpp

Generated for GDAL by doxygen 1.8.8.