My Project

Contents

Chapter 1

Deprecated List

Member OGRLayer::GetInfo (const char ∗)

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 OGRAttrIndex Class Reference

Public Member Functions

- virtual long **GetFirstMatch** (OGRField *psKey)=0
- virtual long * **GetAllMatches** (OGRField *psKey)=0
- virtual long * **GetAllMatches** (OGRField *psKey, long *panFIDList, int *nFIDCount, int *nLength)=0
- virtual OGRErr **AddEntry** (OGRField *psKey, long nFID)=0
- virtual OGRErr **RemoveEntry** (OGRField *psKey, long nFID)=0
- virtual OGRErr **Clear** ()=0

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

- ogr_attrind.h

4.2 OGRDataSource Class Reference

`#include <ogrsf_frmts.h>`

Public Member Functions

- virtual const char * GetName ()=0

  Returns the name of the data source.
- virtual int GetLayerCount ()=0

  Get the number of layers in this data source.
- virtual OGRLayer * GetLayer (int)=0

  Fetch a layer by index.
- virtual OGRLayer * GetLayerByName (const char *)

  Fetch a layer by name.
- virtual OGRErr DeleteLayer (int)

  Delete the indicated layer from the datasource.
- virtual int TestCapability (const char *)=0

  Test if capability is available.
- virtual OGRLayer * CreateLayer (const char *pszName, OGRSpatialReference *poSpatialRef=NULL, OG↩ RwkbGeometryType eGType=wkbUnknown, char **papszOptions=NULL)

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

- virtual OGRLayer * CopyLayer (OGRLayer *poSrcLayer, const char *pszNewName, char **papsz↩ Options=NULL)

  Duplicate an existing layer.
- virtual OGRStyleTable * GetStyleTable ()

  Returns data source style table.
- virtual void SetStyleTableDirectly (OGRStyleTable *poStyleTable)

  Set data source style table.
- virtual void SetStyleTable (OGRStyleTable *poStyleTable)

  Set data source style table.
- virtual OGRLayer * ExecuteSQL (const char *pszStatement, OGRGeometry *poSpatialFilter, const char *pszDialect)

  Execute an SQL statement against the data store.
- virtual void ReleaseResultSet (OGRLayer *poResultsSet)

  Release results of ExecuteSQL().
- virtual OGRErr SyncToDisk ()

  Flush pending changes to disk.
- int Reference ()

  Increment datasource reference count.
- int Dereference ()

  Decrement datasource reference count.
- int GetRefCount () const

  Fetch reference count.
- int GetSummaryRefCount () const

  Fetch reference count of datasource and all owned layers.
- OGRErr Release ()

  Drop a reference to this datasource, and if the reference count drops to zero close (destroy) the datasource.
- OGRSFDriver * GetDriver () const

  Returns the driver that the dataset was opened with.
- void SetDriver (OGRSFDriver *poDriver)

  Sets the driver that the dataset was created or opened with.

Static Public Member Functions

- static void DestroyDataSource (OGRDataSource *)

  Closes opened datasource and releases allocated resources.
- static int **IsGenericSQLDialect** (const char *pszDialect)

Protected Member Functions

- OGRErr **ProcessSQLCreateIndex** (const char *)
- OGRErr **ProcessSQLDropIndex** (const char *)
- OGRErr **ProcessSQLDropTable** (const char *)
- OGRErr **ProcessSQLAlterTableAddColumn** (const char *)
- OGRErr **ProcessSQLAlterTableDropColumn** (const char *)
- OGRErr **ProcessSQLAlterTableAlterColumn** (const char *)
- OGRErr **ProcessSQLAlterTableRenameColumn** (const char *)

Protected Attributes

- OGRStyleTable * **m_poStyleTable**
- int **m_nRefCount**
- OGRSFDriver * **m_poDriver**

Friends

- class **OGRSFDriverRegistrar**

4.2.1 Detailed Description

This class represents a data source. A data source potentially consists of many layers (OGRLayer). A data source normally consists of one, or a related set of files, though the name doesn't have to be a real item in the file system.

When an OGRDataSource is destroyed, all it's associated OGRLayers objects are also destroyed.

4.2.2 Member Function Documentation

4.2.2.1 OGRLayer * OGRDataSource::CopyLayer (OGRLayer * *poSrcLayer*, const char * *pszNewName*, char ** *papszOptions* = NULL) [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 OGR_DS_CopyLayer().

Parameters

poSrcLayer	source layer.
pszNewName	the name of the layer to create.
papszOptions	a StringList of name=value options. Options are driver specific.

Returns

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

4.2.2.2 OGRLayer * OGRDataSource::CreateLayer (const char * *pszName*, OGRSpatialReference * *poSpatialRef* = NULL, OGRwkbGeometryType *eGType* = wkbUnknown, char ** *papszOptions* = NULL) [virtual]

This method attempts to create a new layer on the data source 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.

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.

Example:

```
#include "ogrsf_frmts.h"
#include "cpl_string.h"

...

        OGRLayer *poLayer;
        char     **papszOptions;

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

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

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

4.2.2.3 OGRErr OGRDataSource::DeleteLayer (int *iLayer*) [virtual]

Delete the indicated layer from the datasource.

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

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

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.

4.2.2.4 int OGRDataSource::Dereference ( )

Decrement datasource reference count.

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

Returns

the reference count after decrementing.

4.2.2.5 void OGRDataSource::DestroyDataSource (OGRDataSource * *poDS*) [static]

Closes opened datasource and releases allocated resources.

This static method will close and destroy a datasource. It is equivelent to calling delete on the object, but it ensures that the deallocation is properly executed within the GDAL libraries heap on platforms where this can matter (win32).

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

Parameters

poDS	pointer to allocated datasource object.

4.2.2.6 OGRLayer * OGRDataSource::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 OGRDataSource::ReleaseResultSet() before the data source is closed (destroyed).

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

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

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

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 explicitely 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().

4.2.2.7 OGRSFDriver * OGRDataSource::GetDriver ( ) const

Returns the driver that the dataset was opened with.

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

Returns

NULL if driver info is not available, or pointer to a driver owned by the OGRSFDriverManager.

4.2.2.8 OGRLayer * OGRDataSource::GetLayer (int *iLayer*) [pure virtual]

Fetch a layer by index.

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

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

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.

4.2.2.9 OGRLayer * OGRDataSource::GetLayerByName (const char * *pszLayerName*) [virtual]

Fetch a layer by name.

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

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

Parameters

pszLayerName	the layer name of the layer to fetch.

Returns

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

4.2.2.10 int OGRDataSource::GetLayerCount ( ) [pure virtual]

Get the number of layers in this data source.

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

Returns

layer count.

4.2.2.11 const char * OGRDataSource::GetName ( ) [pure virtual]

Returns the name of the data source.

This string should be sufficient to open the data source if passed to the same OGRSFDriver that this data source was opened with, but it need not be exactly the same string that was used to open the data source. Normally this is a filename.

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

Returns

pointer to an internal name string which should not be modified or freed by the caller.

4.2.2.12 int OGRDataSource::GetRefCount ( ) const

Fetch reference count.

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

Returns

the current reference count for the datasource object itself.

4.2.2.13 void OGRDataSource::GetStyleTable ( ) [virtual]

Returns data source style table.

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

Returns

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

4.2.2.14 int OGRDataSource::GetSummaryRefCount ( ) const

Fetch reference count of datasource and all owned layers.

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

Returns

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

4.2.2.15 int OGRDataSource::Reference ( )

Increment datasource reference count.

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

Returns

the reference count after incrementing.

4.2.2.16 OGRErr OGRDataSource::Release ( )

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

Internally this actually calls the OGRSFDriverRegistrar::ReleaseDataSource() method. This method is essentially a convenient alias.

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

Returns

OGRERR_NONE on success or an error code.

4.2.2.17 void OGRDataSource::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 OGRDataSource. Failure to deallocate a results set before destroying the OGRDataSource may cause errors.

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

Parameters

poResultsSet	the result of a previous ExecuteSQL() call.

4.2.2.18 void OGRDataSource::SetDriver (OGRSFDriver * *poDriver*)

Sets the driver that the dataset was created or opened with.

Note

This method is not exposed as the OGR C API function.

Parameters

poDriver	pointer to driver instance associated with the data source.

4.2.2.19 void OGRDataSource::SetStyleTable (OGRStyleTable * *poStyleTable*) [virtual]

Set data source style table.

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

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

Parameters

poStyleTable	pointer to style table to set

4.2.2.20 void OGRDataSource::SetStyleTableDirectly (OGRStyleTable * *poStyleTable*) [virtual]

Set data source style table.

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

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

Parameters

poStyleTable	pointer to style table to set

4.2.2.21 OGRErr OGRDataSource::SyncToDisk ( ) [virtual]

Flush pending changes to disk.

This call is intended to force the datasource to flush any pending writes to disk, and leave the disk file in a consistent state. It would not normally have any effect on read-only datasources.

Some data sources do not implement this method, and will still return OGRERR_NONE. An error is only returned if an error occurs while attempting to flush to disk.

The default implementation of this method just calls the SyncToDisk() method on each of the layers. Conceptionally, calling SyncToDisk() on a datasource should include any work that might be accomplished by calling SyncToDisk() on layers in that data source.

In any event, you should always close any opened datasource with OGRDataSource::DestroyDataSource() that will ensure all data is correctly flushed.

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

Returns

OGRERR_NONE if no error occurs (even if nothing is done) or an error code.

4.2.2.22 int OGRDataSource::TestCapability (const char * *pszCapability*) [pure virtual]

Test if capability is available.

One of the following data source 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.

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

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

Parameters

pszCapability	the capability to test.

Returns

TRUE if capability available otherwise FALSE.

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

- ogrsf_frmts.h
- ogrsf_frmts.dox

4.3 OGRLayer Class Reference

`#include <ogrsf_frmts.h>`

Public Member Functions

- virtual OGRGeometry * GetSpatialFilter ()

  This method returns the current spatial filter for this layer.
- virtual void SetSpatialFilter (OGRGeometry *)

  Set a new spatial filter.
- virtual void SetSpatialFilterRect (double dfMinX, double dfMinY, double dfMaxX, double dfMaxY)

  Set a new rectangular spatial filter.
- virtual void SetSpatialFilter (int iGeomField, OGRGeometry *)

  Set a new spatial filter.
- virtual void SetSpatialFilterRect (int iGeomField, double dfMinX, double dfMinY, double dfMaxX, double dfMaxY)

  Set a new rectangular spatial filter.
- virtual OGRErr SetAttributeFilter (const char *)

  Set a new attribute query.
- virtual void ResetReading ()=0

  Reset feature reading to start on the first feature.
- virtual OGRFeature * GetNextFeature ()=0

  Fetch the next available feature from this layer.
- virtual OGRErr SetNextByIndex (long nIndex)

  Move read cursor to the nIndex'th feature in the current resultset.
- virtual OGRFeature * GetFeature (long nFID)

  Fetch a feature by its identifier.
- virtual OGRErr SetFeature (OGRFeature *poFeature)

  Rewrite an existing feature.
- virtual OGRErr CreateFeature (OGRFeature *poFeature)

Create and write a new feature within a layer.

- virtual OGRErr DeleteFeature (long nFID)

  Delete feature from layer.
- virtual const char * GetName ()

  Return the layer name.
- virtual OGRwkbGeometryType GetGeomType ()

  Return the layer geometry type.
- virtual OGRFeatureDefn * GetLayerDefn ()=0

  Fetch the schema information for this layer.
- virtual int FindFieldIndex (const char *pszFieldName, int bExactMatch)

  Find the index of field in the layer.
- virtual OGRSpatialReference * GetSpatialRef ()

  Fetch the spatial reference system for this layer.
- virtual int GetFeatureCount (int bForce=TRUE)

  Fetch the feature count in this layer.
- virtual OGRErr GetExtent (OGREnvelope *psExtent, int bForce=TRUE)

  Fetch the extent of this layer.
- virtual OGRErr GetExtent (int iGeomField, OGREnvelope *psExtent, int bForce=TRUE)

  Fetch the extent of this layer, on the specified geometry field.
- virtual int TestCapability (const char *)=0

  Test if this layer supported the named capability.
- virtual const char * GetInfo (const char *)

  Fetch metadata from layer.
- virtual OGRErr CreateField (OGRFieldDefn *poField, int bApproxOK=TRUE)

  Create a new field on a layer.
- virtual OGRErr DeleteField (int iField)

  Delete an existing field on a layer.
- virtual OGRErr ReorderFields (int *panMap)

  Reorder all the fields of a layer.
- virtual OGRErr AlterFieldDefn (int iField, OGRFieldDefn *poNewFieldDefn, int nFlags)

  Alter the definition of an existing field on a layer.
- virtual OGRErr CreateGeomField (OGRGeomFieldDefn *poField, int bApproxOK=TRUE)

  Create a new geometry field on a layer.
- virtual OGRErr SyncToDisk ()

  Flush pending changes to disk.
- virtual OGRStyleTable * GetStyleTable ()

  Returns layer style table.
- virtual void SetStyleTableDirectly (OGRStyleTable *poStyleTable)

  Set layer style table.
- virtual void SetStyleTable (OGRStyleTable *poStyleTable)

  Set layer style table.
- virtual OGRErr **StartTransaction** ()
- virtual OGRErr **CommitTransaction** ()
- virtual OGRErr **RollbackTransaction** ()
- virtual const char * GetFIDColumn ()

  This method returns the name of the underlying database column being used as the FID column, or "" if not supported.
- virtual const char * GetGeometryColumn ()

  This method returns the name of the underlying database column being used as the geometry column, or "" if not supported.
- virtual OGRErr SetIgnoredFields (const char **papszFields)

  Set which fields can be omitted when retrieving features from the layer.

- OGRErr **Intersection** (OGRLayer *pLayerMethod, OGRLayer *pLayerResult, char **papszOptions=NULL, GDALProgressFunc pfnProgress=NULL, void *pProgressArg=NULL)
- OGRErr **Union** (OGRLayer *pLayerMethod, OGRLayer *pLayerResult, char **papszOptions=NULL, GDALProgressFunc pfnProgress=NULL, void *pProgressArg=NULL)
- OGRErr **SymDifference** (OGRLayer *pLayerMethod, OGRLayer *pLayerResult, char **papszOptions, GDALProgressFunc pfnProgress, void *pProgressArg)
- OGRErr **Identity** (OGRLayer *pLayerMethod, OGRLayer *pLayerResult, char **papszOptions=NULL, GDALProgressFunc pfnProgress=NULL, void *pProgressArg=NULL)
- OGRErr **Update** (OGRLayer *pLayerMethod, OGRLayer *pLayerResult, char **papszOptions=NULL, GDALProgressFunc pfnProgress=NULL, void *pProgressArg=NULL)
- OGRErr **Clip** (OGRLayer *pLayerMethod, OGRLayer *pLayerResult, char **papszOptions=NULL, GDALProgressFunc pfnProgress=NULL, void *pProgressArg=NULL)
- OGRErr **Erase** (OGRLayer *pLayerMethod, OGRLayer *pLayerResult, char **papszOptions=NULL, GDALProgressFunc pfnProgress=NULL, void *pProgressArg=NULL)
- int Reference ()

  Increment layer reference count.
- int Dereference ()

  Decrement layer reference count.
- int GetRefCount () const

  Fetch reference count.
- GIntBig **GetFeaturesRead** ()
- OGRErr ReorderField (int iOldFieldPos, int iNewFieldPos)

  Reorder an existing field on a layer.
- int **AttributeFilterEvaluationNeedsGeometry** ()
- OGRErr **InitializeIndexSupport** (const char *)
- OGRLayerAttrIndex * **GetIndex** ()

Protected Member Functions

- int **FilterGeometry** (OGRGeometry *)
- int **InstallFilter** (OGRGeometry *)
- OGRErr **GetExtentInternal** (int iGeomField, OGREnvelope *psExtent, int bForce)

Protected Attributes

- int **m_bFilterIsEnvelope**
- OGRGeometry * **m_poFilterGeom**
- OGRPreparedGeometry * **m_pPreparedFilterGeom**
- OGREnvelope **m_sFilterEnvelope**
- int **m_iGeomFieldFilter**
- OGRStyleTable * **m_poStyleTable**
- OGRFeatureQuery * **m_poAttrQuery**
- char * **m_pszAttrQueryString**
- OGRLayerAttrIndex * **m_poAttrIndex**
- int **m_nRefCount**
- GIntBig **m_nFeaturesRead**

4.3.1 Detailed Description

This class represents a layer of simple features, with access methods.

4.3.2 Member Function Documentation

4.3.2.1 OGRErr OGRLayer::AlterFieldDefn (int *iField,* OGRFieldDefn * *poNewFieldDefn,* int *nFlags*) [virtual]

Alter the definition of an existing field on a layer.

You must use this to alter the definition of an existing field of a real layer. Internally the OGRFeatureDefn for the layer will be updated to reflect the altered field. Applications should never modify the OGRFeatureDefn used by a layer directly.

This method should not be called while there are feature objects in existance that were obtained or created with the previous layer definition.

Not all drivers support this method. You can query a layer to check if it supports it with the OLCAlterFieldDefn capability. Some drivers may only support this method while there are still no features in the layer. When it is supported, the existings features of the backing file/database should be updated accordingly. Some drivers might also not support all update flags.

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

Parameters

iField	index of the field whose definition must be altered.
poNewFieldDefn	new field definition
nFlags	combination of ALTER_NAME_FLAG, ALTER_TYPE_FLAG and ALTER_WIDTH_PRECI↩ SION_FLAG to indicate which of the name and/or type and/or width and precision fields from the new field definition must be taken into account.

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

4.3.2.2 OGRErr OGRLayer::CreateFeature (OGRFeature * *poFeature*) [virtual]

Create and write a new feature within a layer.

The passed feature is written to the layer as a new feature, rather than overwriting an existing one. If the feature has a feature id other than OGRNullFID, then the native implementation may use that as the feature id of the new feature, but not necessarily. Upon successful return the passed feature will have been updated with the new feature id.

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

Parameters

poFeature	the feature to write to disk.

Returns

OGRERR_NONE on success.

4.3.2.3 OGRErr OGRLayer::CreateField (OGRFieldDefn * *poField,* int *bApproxOK* = TRUE) [virtual]

Create a new field on a layer.

You must use this to create new fields on a real layer. Internally the OGRFeatureDefn for the layer will be updated to reflect the new field. Applications should never modify the OGRFeatureDefn used by a layer directly.

This method should not be called while there are feature objects in existance that were obtained or created with the previous layer definition.

Not all drivers support this method. You can query a layer to check if it supports it with the OLCCreateField capability. Some drivers may only support this method while there are still no features in the layer. When it is supported, the existings features of the backing file/database should be updated accordingly.

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

Parameters

poField	field definition to write to disk.
bApproxOK	If TRUE, the field may be created in a slightly different form depending on the limitations of the format driver.

Returns

OGRERR_NONE on success.

4.3.2.4 OGRErr OGRLayer::CreateGeomField (OGRGeomFieldDefn * *poField,* int *bApproxOK* = TRUE) [virtual]

Create a new geometry field on a layer.

You must use this to create new geometry fields on a real layer. Internally the OGRFeatureDefn for the layer will be updated to reflect the new field. Applications should never modify the OGRFeatureDefn used by a layer directly.

This method should not be called while there are feature objects in existance that were obtained or created with the previous layer definition.

Not all drivers support this method. You can query a layer to check if it supports it with the OLCCreateGeomField capability. Some drivers may only support this method while there are still no features in the layer. When it is supported, the existings features of the backing file/database should be updated accordingly.

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

Parameters

poField	geometry field definition to write to disk.
bApproxOK	If TRUE, the field may be created in a slightly different form depending on the limitations of the format driver.

Returns

OGRERR_NONE on success.

Since

OGR 1.11

4.3.2.5 OGRErr OGRLayer::DeleteFeature (long *nFID*) [virtual]

Delete feature from layer.

The feature with the indicated feature id is deleted from the layer if supported by the driver. Most drivers do not support feature deletion, and will return OGRERR_UNSUPPORTED_OPERATION. The TestCapability() layer method may be called with OLCDeleteFeature to check if the driver supports feature deletion.

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

Parameters

nFID	the feature id to be deleted from the layer

Returns

OGRERR_NONE on success.

4.3.2.6 OGRErr OGRLayer::DeleteField (int *iField*) [virtual]

Delete an existing field on a layer.

You must use this to delete existing fields on a real layer. Internally the OGRFeatureDefn for the layer will be updated to reflect the deleted field. Applications should never modify the OGRFeatureDefn used by a layer directly.

This method should not be called while there are feature objects in existance that were obtained or created with the previous layer definition.

Not all drivers support this method. You can query a layer to check if it supports it with the OLCDeleteField capability. Some drivers may only support this method while there are still no features in the layer. When it is supported, the existings features of the backing file/database should be updated accordingly.

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

Parameters

iField	index of the field to delete.

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

4.3.2.7 int OGRLayer::Dereference ( )

Decrement layer reference count.

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

Returns

the reference count after decrementing.

4.3.2.8 int OGRLayer::FindFieldIndex (const char * *pszFieldName,* int *bExactMatch*) [virtual]

Find the index of field in the layer.

The returned number is the index of the field in the layers, or -1 if the field doesn't exist.

If bExactMatch is set to FALSE and the field doesn't exists in the given form the driver might apply some changes to make it match, like those it might do if the layer was created (eg. like LAUNDER in the OCI driver).

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

Returns

field index, or -1 if the field doesn't exist

4.3.2.9 OGRErr OGRLayer::GetExtent (OGREnvelope * *psExtent*, int *bForce* = TRUE) [virtual]

Fetch the extent of this layer.

Returns the extent (MBR) of the data in the layer. If bForce is FALSE, and it would be expensive to establish the extent then OGRERR_FAILURE will be returned indicating that the extent isn't know. If bForce is TRUE then some implementations will actually scan the entire layer once to compute the MBR of all the features in the layer.

Depending on the drivers, the returned extent may or may not take the spatial filter into account. So it is safer to call GetExtent() without setting a spatial filter.

Layers without any geometry may return OGRERR_FAILURE just indicating that no meaningful extents could be collected.

Note that some implementations of this method may alter the read cursor of the layer.

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

Parameters

psExtent	the structure in which the extent value will be returned.
bForce	Flag indicating whether the extent should be computed even if it is expensive.

Returns

OGRERR_NONE on success, OGRERR_FAILURE if extent not known.

4.3.2.10 OGRErr OGRLayer::GetExtent (int *iGeomField*, OGREnvelope * *psExtent*, int *bForce* = TRUE) [virtual]

Fetch the extent of this layer, on the specified geometry field.

Returns the extent (MBR) of the data in the layer. If bForce is FALSE, and it would be expensive to establish the extent then OGRERR_FAILURE will be returned indicating that the extent isn't know. If bForce is TRUE then some implementations will actually scan the entire layer once to compute the MBR of all the features in the layer.

Depending on the drivers, the returned extent may or may not take the spatial filter into account. So it is safer to call GetExtent() without setting a spatial filter.

Layers without any geometry may return OGRERR_FAILURE just indicating that no meaningful extents could be collected.

Note that some implementations of this method may alter the read cursor of the layer.

Note to driver implementators: if you implement GetExtent(int,OGREnvelope*,int), you must also implement GetExtent(OGREnvelope*, int) to make it call GetExtent(0,OGREnvelope*,int).

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

Parameters

iGeomField	the index of the geometry field on which to compute the extent.
psExtent	the structure in which the extent value will be returned.
bForce	Flag indicating whether the extent should be computed even if it is expensive.

Returns

OGRERR_NONE on success, OGRERR_FAILURE if extent not known.

4.3.2.11 OGRFeature * OGRLayer::GetFeature (long *nFID*) [virtual]

Fetch a feature by its identifier.

This function will attempt to read the identified feature. The nFID value cannot be OGRNullFID. Success or failure of this operation is unaffected by the spatial or attribute filters (and specialized implementations in drivers should make sure that they do not take into account spatial or attribute filters).

If this method returns a non-NULL feature, it is guaranteed that its feature id (OGRFeature::GetFID()) will be the same as nFID.

Use OGRLayer::TestCapability(OLCRandomRead) to establish if this layer supports efficient random access reading via GetFeature(); however, the call should always work if the feature exists as a fallback implementation just scans all the features in the layer looking for the desired feature.

Sequential reads (with GetNextFeature()) are generally considered interrupted by a GetFeature() call.

The returned feature should be free with OGRFeature::DestroyFeature().

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

Parameters

nFID	the feature id of the feature to read.

Returns

a feature now owned by the caller, or NULL on failure.

4.3.2.12 int OGRLayer::GetFeatureCount (int *bForce* = TRUE) [virtual]

Fetch the feature count in this layer.

Returns the number of features in the layer. For dynamic databases the count may not be exact. If bForce is FALSE, and it would be expensive to establish the feature count a value of -1 may be returned indicating that the count isn't know. If bForce is TRUE some implementations will actually scan the entire layer once to count objects.

The returned count takes the spatial filter into account.

Note that some implementations of this method may alter the read cursor of the layer.

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

Parameters

bForce	Flag indicating whether the count should be computed even if it is expensive.

Returns

feature count, -1 if count not known.

4.3.2.13 const char * OGRLayer::GetFIDColumn ( ) [virtual]

This method returns the name of the underlying database column being used as the FID column, or "" if not supported.

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

Returns

fid column name.

4.3.2.14 const char * OGRLayer::GetGeometryColumn ( ) [virtual]

This method returns the name of the underlying database column being used as the geometry column, or "" if not supported.

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

Returns

geometry column name.

4.3.2.15 OGRwkbGeometryType OGRLayer::GetGeomType ( ) [virtual]

Return the layer geometry type.

This returns the same result as GetLayerDefn()->GetGeomType(), but for a few drivers, calling GetGeomType() directly can avoid lengthy layer definition initialization.

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

If this method is derived in a driver, it must be done such that it returns the same content as GetLayerDefn()->GetGeomType().

Returns

the geometry type

Since

OGR 1.8.0

4.3.2.16 const char * OGRLayer::GetInfo (const char * *pszTag*) [virtual]

Fetch metadata from layer.

This method can be used to fetch various kinds of metadata or layer specific information encoded as a string. It is anticipated that various tag values will be defined with well known semantics, while other tags will be used for driver/application specific purposes.

This method is deprecated and will be replaced with a more general metadata model in the future. At this time no drivers return information via the GetInfo() call.

Parameters

pszTag	the tag for which information is being requested.

Returns

the value of the requested tag, or NULL if that tag does not have a value, or is unknown.

Deprecated

4.3.2.17 OGRFeatureDefn * OGRLayer::GetLayerDefn ( ) [pure virtual]

Fetch the schema information for this layer.

The returned OGRFeatureDefn is owned by the OGRLayer, and should not be modified or freed by the application. It encapsulates the attribute schema of the features of the layer.

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

Returns

feature definition.

4.3.2.18 const char * OGRLayer::GetName ( ) [virtual]

Return the layer name.

This returns the same content as GetLayerDefn()->GetName(), but for a few drivers, calling GetName() directly can avoid lengthy layer definition initialization.

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

If this method is derived in a driver, it must be done such that it returns the same content as GetLayerDefn()->GetName().

Returns

the layer name (must not been freed)

Since

OGR 1.8.0

4.3.2.19 OGRFeature * OGRLayer::GetNextFeature ( ) [pure virtual]

Fetch the next available feature from this layer.

The returned feature becomes the responsiblity of the caller to delete with OGRFeature::DestroyFeature(). It is critical that all features associated with an OGRLayer (more specifically an OGRFeatureDefn) be deleted before that layer/datasource is deleted.

Only features matching the current spatial filter (set with SetSpatialFilter()) will be returned.

This method implements sequential access to the features of a layer. The ResetReading() method can be used to start at the beginning again.

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

Returns

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

4.3.2.20 int OGRLayer::GetRefCount ( ) const

Fetch reference count.

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

Returns

the current reference count for the layer object itself.

4.3.2.21 OGRGeometry * OGRLayer::GetSpatialFilter ( ) [virtual]

This method returns the current spatial filter for this layer.

The returned pointer is to an internally owned object, and should not be altered or deleted by the caller.

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

Returns

spatial filter geometry.

4.3.2.22 OGRSpatialReference * OGRLayer::GetSpatialRef ( ) [virtual]

Fetch the spatial reference system for this layer.

The returned object is owned by the OGRLayer and should not be modified or freed by the application.

Starting with OGR 1.11, several geometry fields can be associated to a feature definition. Each geometry field can have its own spatial reference system, which is returned by OGRGeomFieldDefn::GetSpatialRef(). OGRLayer::↩ GetSpatialRef() is equivalent to GetLayerDefn()->GetGeomFieldDefn(0)->GetSpatialRef()

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

Returns

spatial reference, or NULL if there isn't one.

4.3.2.23 void OGRLayer::GetStyleTable ( ) [virtual]

Returns layer style table.

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

Returns

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

4.3.2.24 int OGRLayer::Reference ( )

Increment layer reference count.

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

Returns

the reference count after incrementing.

4.3.2.25 OGRErr OGRLayer::ReorderField (int *iOldFieldPos,* int *iNewFieldPos*)

Reorder an existing field on a layer.

This method is a conveniency wrapper of ReorderFields() dedicated to move a single field. It is a non-virtual method, so drivers should implement ReorderFields() instead.

You must use this to reorder existing fields on a real layer. Internally the OGRFeatureDefn for the layer will be updated to reflect the reordering of the fields. Applications should never modify the OGRFeatureDefn used by a layer directly.

This method should not be called while there are feature objects in existance that were obtained or created with the previous layer definition.

The field definition that was at initial position iOldFieldPos will be moved at position iNewFieldPos, and elements between will be shuffled accordingly.

For example, let suppose the fields were "0","1","2","3","4" initially. ReorderField(1, 3) will reorder them as "0","2","3","1","4".

Not all drivers support this method. You can query a layer to check if it supports it with the OLCReorderFields capability. Some drivers may only support this method while there are still no features in the layer. When it is supported, the existings features of the backing file/database should be updated accordingly.

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

Parameters

iOldFieldPos	previous position of the field to move. Must be in the range [0,GetFieldCount()-1].
iNewFieldPos	new position of the field to move. Must be in the range [0,GetFieldCount()-1].

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

4.3.2.26 OGRErr OGRLayer::ReorderFields (int * *panMap*) [virtual]

Reorder all the fields of a layer.

You must use this to reorder existing fields on a real layer. Internally the OGRFeatureDefn for the layer will be updated to reflect the reordering of the fields. Applications should never modify the OGRFeatureDefn used by a layer directly.

This method should not be called while there are feature objects in existance that were obtained or created with the previous layer definition.

panMap is such that,for each field definition at position i after reordering, its position before reordering was panMap[i].

For example, let suppose the fields were "0","1","2","3","4" initially. ReorderFields([0,2,3,1,4]) will reorder them as "0","2","3","1","4".

Not all drivers support this method. You can query a layer to check if it supports it with the OLCReorderFields capability. Some drivers may only support this method while there are still no features in the layer. When it is supported, the existings features of the backing file/database should be updated accordingly.

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

Parameters

panMap	an array of GetLayerDefn()->GetFieldCount() elements which is a permutation of [0, GetLayerDefn()->GetFieldCount()-1].

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

4.3.2.27 void OGRLayer::ResetReading ( ) [pure virtual]

Reset feature reading to start on the first feature.

This affects GetNextFeature().

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

4.3.2.28 void OGRLayer::SetAttributeFilter (const char * *pszQuery*) [virtual]

Set a new attribute query.

This method sets the attribute query string to be used when fetching features via the GetNextFeature() method. Only features for which the query evaluates as true will be returned.

The query string should be in the format of an SQL WHERE clause. For instance "population > 1000000 and population < 5000000" where population is an attribute in the layer. The query format is normally a restricted form of SQL WHERE clause as described in the "WHERE" section of the OGR SQL tutorial. In some cases (RDBMS backed drivers) the native capabilities of the database may be used to interprete the WHERE clause in which case the capabilities will be broader than those of OGR SQL.

Note that installing a query string will generally result in resetting the current reading position (ala ResetReading()).

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

Parameters

pszQuery	query in restricted SQL WHERE format, or NULL to clear the current query.

Returns

OGRERR_NONE if successfully installed, or an error code if the query expression is in error, or some other failure occurs.

4.3.2.29 OGRErr OGRLayer::SetFeature (OGRFeature * *poFeature*) [virtual]

Rewrite an existing feature.

This method will write a feature to the layer, based on the feature id within the OGRFeature.

Use OGRLayer::TestCapability(OLCRandomWrite) to establish if this layer supports random access writing via SetFeature().

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

Parameters

poFeature	the feature to write.

Returns

OGRERR_NONE if the operation works, otherwise an appropriate error code.

4.3.2.30 OGRErr OGRLayer::SetIgnoredFields (const char ** *papszFields*) [virtual]

Set which fields can be omitted when retrieving features from the layer.

If the driver supports this functionality (testable using OLCIgnoreFields capability), it will not fetch the specified fields in subsequent calls to GetFeature() / GetNextFeature() and thus save some processing time and/or bandwidth.

Besides field names of the layers, the following special fields can be passed: "OGR_GEOMETRY" to ignore geometry and "OGR_STYLE" to ignore layer style.

By default, no fields are ignored.

This method is the same as the C function OGR_L_SetIgnoredFields()

Parameters

papszFields	an array of field names terminated by NULL item. If NULL is passed, the ignored list is cleared.

Returns

OGRERR_NONE if all field names have been resolved (even if the driver does not support this method)

4.3.2.31 OGRErr OGRLayer::SetNextByIndex (long *nIndex*) [virtual]

Move read cursor to the nIndex'th feature in the current resultset.

This method allows positioning of a layer such that the GetNextFeature() call will read the requested feature, where nIndex is an absolute index into the current result set. So, setting it to 3 would mean the next feature read with GetNextFeature() would have been the 4th feature to have been read if sequential reading took place from the beginning of the layer, including accounting for spatial and attribute filters.

Only in rare circumstances is SetNextByIndex() efficiently implemented. In all other cases the default implementation which calls ResetReading() and then calls GetNextFeature() nIndex times is used. To determine if fast seeking is available on the current layer use the TestCapability() method with a value of OLCFastSetNextByIndex.

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

Parameters

nIndex	the index indicating how many steps into the result set to seek.

Returns

OGRERR_NONE on success or an error code.

4.3.2.32 void OGRLayer::SetSpatialFilter (OGRGeometry * *poFilter*) [virtual]

Set a new spatial filter.

This method set the geometry to be used as a spatial filter when fetching features via the GetNextFeature() method. Only features that geometrically intersect the filter geometry will be returned.

Currently this test is may be inaccurately implemented, but it is guaranteed that all features who's envelope (as returned by OGRGeometry::getEnvelope()) overlaps the envelope of the spatial filter will be returned. This can result in more shapes being returned that should strictly be the case.

This method makes an internal copy of the passed geometry. The passed geometry remains the responsibility of the caller, and may be safely destroyed.

For the time being the passed filter geometry should be in the same SRS as the layer (as returned by OGRLayer::GetSpatialRef()). In the future this may be generalized.

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

Parameters

poFilter	the geometry to use as a filtering region. NULL may be passed indicating that the current spatial filter should be cleared, but no new one instituted.

4.3.2.33 void OGRLayer::SetSpatialFilter (int *iGeomField*, OGRGeometry * *poFilter*) [virtual]

Set a new spatial filter.

This method set the geometry to be used as a spatial filter when fetching features via the GetNextFeature() method. Only features that geometrically intersect the filter geometry will be returned.

Currently this test is may be inaccurately implemented, but it is guaranteed that all features who's envelope (as returned by OGRGeometry::getEnvelope()) overlaps the envelope of the spatial filter will be returned. This can result in more shapes being returned that should strictly be the case.

This method makes an internal copy of the passed geometry. The passed geometry remains the responsibility of the caller, and may be safely destroyed.

For the time being the passed filter geometry should be in the same SRS as the geometry field definition it corresponds to (as returned by GetLayerDefn()->GetGeomFieldDefn(iGeomField)->GetSpatialRef()). In the future this may be generalized.

Note that only the last spatial filter set is applied, even if several successive calls are done with different iGeomField values.

Note to driver implementators: if you implement SetSpatialFilter(int,OGRGeometry*), you must also implement SetSpatialFilter(OGRGeometry*) to make it call SetSpatialFilter(0,OGRGeometry*).

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

Parameters

iGeomField	index of the geometry field on which the spatial filter operates.
poFilter	the geometry to use as a filtering region. NULL may be passed indicating that the current spatial filter should be cleared, but no new one instituted.

Since

GDAL 1.11

4.3.2.34 void OGRLayer::SetSpatialFilterRect (double *dfMinX*, double *dfMinY*, double *dfMaxX*, double *dfMaxY*) [virtual]

Set a new rectangular spatial filter.

This method set rectangle to be used as a spatial filter when fetching features via the GetNextFeature() method. Only features that geometrically intersect the given rectangle will be returned.

The x/y values should be in the same coordinate system as the layer as a whole (as returned by OGRLayer::GetSpatialRef()). Internally this method is normally implemented as creating a 5 vertex closed rectangular polygon and passing it to OGRLayer::SetSpatialFilter(). It exists as a convenience.

The only way to clear a spatial filter set with this method is to call OGRLayer::SetSpatialFilter(NULL).

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

Parameters

dfMinX	the minimum X coordinate for the rectangular region.
dfMinY	the minimum Y coordinate for the rectangular region.
dfMaxX	the maximum X coordinate for the rectangular region.
dfMaxY	the maximum Y coordinate for the rectangular region.

4.3.2.35 void OGRLayer::SetSpatialFilterRect (int *iGeomField*, double *dfMinX*, double *dfMinY*, double *dfMaxX*, double *dfMaxY*) [virtual]

Set a new rectangular spatial filter.

This method set rectangle to be used as a spatial filter when fetching features via the GetNextFeature() method. Only features that geometrically intersect the given rectangle will be returned.

The x/y values should be in the same coordinate system as as the geometry field definition it corresponds to (as returned by GetLayerDefn()->GetGeomFieldDefn(iGeomField)->GetSpatialRef()). Internally this method is normally implemented as creating a 5 vertex closed rectangular polygon and passing it to OGRLayer::SetSpatialFilter(). It exists as a convenience.

The only way to clear a spatial filter set with this method is to call OGRLayer::SetSpatialFilter(NULL).

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

Parameters

iGeomField	index of the geometry field on which the spatial filter operates.
dfMinX	the minimum X coordinate for the rectangular region.
dfMinY	the minimum Y coordinate for the rectangular region.
dfMaxX	the maximum X coordinate for the rectangular region.
dfMaxY	the maximum Y coordinate for the rectangular region.

Since

GDAL 1.11

4.3.2.36 void OGRLayer::SetStyleTable (OGRStyleTable * *poStyleTable*) [virtual]

Set layer style table.

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

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

Parameters

poStyleTable	pointer to style table to set

4.3.2.37 void OGRLayer::SetStyleTableDirectly (OGRStyleTable * *poStyleTable*) [virtual]

Set layer style table.

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

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

Parameters

poStyleTable	pointer to style table to set

4.3.2.38 OGRErr OGRLayer::SyncToDisk ( ) [virtual]

Flush pending changes to disk.

This call is intended to force the layer to flush any pending writes to disk, and leave the disk file in a consistent state. It would not normally have any effect on read-only datasources.

Some layers do not implement this method, and will still return OGRERR_NONE. The default implementation just returns OGRERR_NONE. An error is only returned if an error occurs while attempting to flush to disk.

In any event, you should always close any opened datasource with OGRDataSource::DestroyDataSource() that will ensure all data is correctly flushed.

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

Returns

OGRERR_NONE if no error occurs (even if nothing is done) or an error code.

4.3.2.39 int OGRLayer::TestCapability (const char * *pszCap*) [pure virtual]

Test if this layer supported the named capability.

The capability codes that can be tested are represented as strings, but #defined constants exists to ensure correct spelling. Specific layer types may implement class specific capabilities, but this can't generally be discovered by the caller.

- **OLCRandomRead** / "RandomRead": TRUE if the GetFeature() method is implemented in an optimized way for this layer, as opposed to the default implementation using ResetReading() and GetNextFeature() to find the requested feature id.
- **OLCSequentialWrite** / "SequentialWrite": TRUE if the CreateFeature() method works for this layer. Note this means that this particular layer is writable. The same OGRLayer class may returned FALSE for other layer instances that are effectively read-only.
- **OLCRandomWrite** / "RandomWrite": TRUE if the SetFeature() method is operational on this layer. Note this means that this particular layer is writable. The same OGRLayer class may returned FALSE for other layer instances that are effectively read-only.
- **OLCFastSpatialFilter** / "FastSpatialFilter": TRUE if this layer implements spatial filtering efficiently. Layers that effectively read all features, and test them with the OGRFeature intersection methods should return FALSE. This can be used as a clue by the application whether it should build and maintain its own spatial index for features in this layer.
- **OLCFastFeatureCount** / "FastFeatureCount": TRUE if this layer can return a feature count (via GetFeatureCount()) efficiently ... ie. without counting the features. In some cases this will return TRUE until a spatial filter is installed after which it will return FALSE.
- **OLCFastGetExtent** / "FastGetExtent": TRUE if this layer can return its data extent (via GetExtent()) efficiently ... ie. without scanning all the features. In some cases this will return TRUE until a spatial filter is installed after which it will return FALSE.
- **OLCFastSetNextByIndex** / "FastSetNextByIndex": TRUE if this layer can perform the SetNextByIndex() call efficiently, otherwise FALSE.
- **OLCCreateField** / "CreateField": TRUE if this layer can create new fields on the current layer using CreateField(), otherwise FALSE.
- **OLCCreateGeomField** / "CreateGeomField": (GDAL >= 1.11) TRUE if this layer can create new geometry fields on the current layer using CreateGeomField(), otherwise FALSE.
- **OLCDeleteField** / "DeleteField": TRUE if this layer can delete existing fields on the current layer using DeleteField(), otherwise FALSE.
- **OLCReorderFields** / "ReorderFields": TRUE if this layer can reorder existing fields on the current layer using ReorderField() or ReorderFields(), otherwise FALSE.
- **OLCAlterFieldDefn** / "AlterFieldDefn": TRUE if this layer can alter the definition of an existing field on the current layer using AlterFieldDefn(), otherwise FALSE.
- **OLCDeleteFeature** / "DeleteFeature": TRUE if the DeleteFeature() method is supported on this layer, otherwise FALSE.
- **OLCStringsAsUTF8** / "StringsAsUTF8": TRUE if values of OFTString fields are assured to be in UTF-8 format. If FALSE the encoding of fields is uncertain, though it might still be UTF-8.
- **OLCTransactions** / "Transactions": TRUE if the StartTransaction(), CommitTransaction() and RollbackTransaction() methods work in a meaningful way, otherwise FALSE.

- **OLCIgnoreFields** / "IgnoreFields": TRUE if fields, geometry and style will be omitted when fetching features as set by SetIgnoredFields() method.

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

Parameters

pszCap	the name of the capability to test.

Returns

TRUE if the layer has the requested capability, or FALSE otherwise. OGRLayers will return FALSE for any unrecognised capabilities.

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

- ogrsf_frmts.h
- ogrsf_frmts.dox

4.4 OGRLayerAttrIndex Class Reference

Public Member Functions

- virtual OGRErr **Initialize** (const char *pszIndexPath, OGRLayer *)=0
- virtual OGRErr **CreateIndex** (int iField)=0
- virtual OGRErr **DropIndex** (int iField)=0
- virtual OGRErr **IndexAllFeatures** (int iField=-1)=0
- virtual OGRErr **AddToIndex** (OGRFeature *poFeature, int iField=-1)=0
- virtual OGRErr **RemoveFromIndex** (OGRFeature *poFeature)=0
- virtual OGRAttrIndex * **GetFieldIndex** (int iField)=0

Protected Attributes

- OGRLayer * **poLayer**
- char * **pszIndexPath**

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

- ogr_attrind.h

4.5 OGRSFDriver Class Reference

`#include <ogrsf_frmts.h>`

Public Member Functions

- virtual const char * GetName ()=0

  Fetch name of driver (file format). This name should be relatively short (10-40 characters), and should reflect the underlying file format. For instance "ESRI Shapefile".
- virtual OGRDataSource * Open (const char *pszName, int bUpdate=FALSE)=0

  Attempt to open file with this driver.
- virtual int TestCapability (const char *)=0

  Test if capability is available.
- virtual OGRDataSource * CreateDataSource (const char *pszName, char **=NULL)

  This method attempts to create a new data source based on the passed driver.
- virtual OGRErr DeleteDataSource (const char *pszName)

Delete a datasource.

- virtual OGRDataSource * CopyDataSource (OGRDataSource *poSrcDS, const char *pszNewName, char **papszOptions=NULL)

  This method creates a new datasource by copying all the layers from the source datasource.

4.5.1 Detailed Description

Represents an operational format driver.

One OGRSFDriver derived class will normally exist for each file format registered for use, regardless of whether a file has or will be opened. The list of available drivers is normally managed by the OGRSFDriverRegistrar.

4.5.2 Member Function Documentation

4.5.2.1 OGRDataSource * OGRSFDriver::CopyDataSource (OGRDataSource * *poSrcDS*, const char * *pszNewName*, char ** *papszOptions* = NULL) [virtual]

This method creates a new datasource by copying all the layers from the source datasource.

It is important to call OGRDataSource::DestroyDataSource() when the datasource is no longer used to ensure that all data has been properly flushed to disk.

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

Parameters

poSrcDS	source datasource
pszNewName	the name for the new data source. UTF-8 encoded.
papszOptions	a StringList of name=value options. Options are driver specific, and driver information can be found at the following url: http://www.gdal.org/ogr/ogr_formats.html

Returns

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

4.5.2.2 OGRDataSource * OGRSFDriver::CreateDataSource (const char * *pszName*, char ** *papszOptions* = NULL) [virtual]

This method attempts to create a new data source based on the passed driver.

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

It is important to call OGRDataSource::DestroyDataSource() when the datasource is no longer used to ensure that all data has been properly flushed to disk.

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

Note

This method does **NOT** attach driver instance to the returned data source, so caller should expect that OGRDataSource::GetDriver() will return NULL pointer. In order to attach driver to the returned data source, it is required to use C function OGR_Dr_CreateDataSource. This behavior is related to fix of issue reported in Ticket #1233.

Parameters

pszName	the name for the new data source. UTF-8 encoded.
papszOptions	a StringList of name=value options. Options are driver specific, and driver information can be found at the following url: http://www.gdal.org/ogr/ogr_formats.html

Returns

NULL is returned on failure, or a new OGRDataSource on success.

4.5.2.3 OGRErr OGRSFDriver::DeleteDataSource (const char * *pszDataSource*) [virtual]

Delete a datasource.

Delete (from the disk, in the database, ...) the named datasource. Normally it would be safest if the datasource was not open at the time.

Whether this is a supported operation on this driver case be tested using TestCapability() on ODrCDeleteDataSource.

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

Parameters

pszDataSource	the name of the datasource to delete.

Returns

OGRERR_NONE on success, and OGRERR_UNSUPPORTED_OPERATION if this is not supported by this driver.

4.5.2.4 const char * OGRSFDriver::GetName ( ) [pure virtual]

Fetch name of driver (file format). This name should be relatively short (10-40 characters), and should reflect the underlying file format. For instance "ESRI Shapefile".

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

Returns

driver name. This is an internal string and should not be modified or freed.

4.5.2.5 OGRDataSource * OGRSFDriver::Open (const char * *pszName*, int *bUpdate* = FALSE) [pure virtual]

Attempt to open file with this driver.

This method is what OGRSFDriverRegistrar uses to implement its Open() method. See it for more details.

Note, drivers do not normally set their own m_poDriver value, so a direct call to this method (instead of indirectly via OGRSFDriverRegistrar) will usually result in a datasource that does not know what driver it relates to if GetDriver() is called on the datasource. The application may directly call SetDriver() after opening with this method to avoid this problem.

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())

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

Parameters

pszName	the name of the file, or data source to try and open.
bUpdate	TRUE if update access is required, otherwise FALSE (the default).

Returns

NULL on error or if the pass name is not supported by this driver, otherwise a pointer to an OGRDataSource. This OGRDataSource should be closed by deleting the object when it is no longer needed.

4.5.2.6 int OGRSFDriver::TestCapability (const char * *pszCapability*) [pure virtual]

Test if capability is available.

One of the following data source 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.

- **ODrCCreateDataSource**: True if this driver can support creating data sources.
- **ODrCDeleteDataSource**: True if this driver supports deleting data sources.

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

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

Parameters

pszCapability	the capability to test.

Returns

TRUE if capability available otherwise FALSE.

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

- ogrsf_frmts.h
- ogrsf_frmts.dox

4.6 OGRSFDriverRegistrar Class Reference

`#include <ogrsf_frmts.h>`

Public Member Functions

- OGRDataSource * **OpenShared** (const char *pszName, int bUpdate=FALSE, OGRSFDriver **ppoDriver=NULL)
- OGRErr **ReleaseDataSource** (OGRDataSource *)
- void RegisterDriver (OGRSFDriver *poDriver)

  Add a driver to the list of registered drivers.
- void DeregisterDriver (OGRSFDriver *poDriver)

  Remove the passed driver from the list of registered drivers.
- int GetDriverCount (void)

  Fetch the number of registered drivers.
- OGRSFDriver * GetDriver (int iDriver)

Fetch the indicated driver.

- OGRSFDriver * GetDriverByName (const char *)

  Fetch the indicated driver.

- int GetOpenDSCount ()

  Return the number of opened datasources.

- OGRDataSource * GetOpenDS (int)

  Return the iDS th datasource opened.

- void **AutoLoadDrivers** ()

Static Public Member Functions

- static OGRSFDriverRegistrar * GetRegistrar ()

  Return the driver manager, creating one if none exist.

- static OGRDataSource * Open (const char *pszName, int bUpdate=FALSE, OGRSFDriver **ppoDriver=NULL)

  Open a file / data source with one of the registered drivers.

4.6.1 Detailed Description

Singleton manager for OGRSFDriver instances that will be used to try and open datasources. Normally the registrar is populated with standard drivers using the OGRRegisterAll() function and does not need to be directly accessed. The driver registrar and all registered drivers may be cleaned up on shutdown using OGRCleanupAll().

4.6.2 Member Function Documentation

4.6.2.1 void OGRSFDriverRegistrar::DeregisterDriver (OGRSFDriver * *poDriver*)

Remove the passed driver from the list of registered drivers.

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

Parameters

poDriver	the driver to deregister.

Since

GDAL 1.8.0

4.6.2.2 OGRSFDriver * OGRSFDriverRegistrar::GetDriver (int *iDriver*)

Fetch the indicated driver.

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

Parameters

iDriver	the driver index, from 0 to GetDriverCount()-1.

Returns

the driver, or NULL if iDriver is out of range.

4.6.2.3 OGRSFDriver * OGRSFDriverRegistrar::GetDriverByName (const char * *pszName*)

Fetch the indicated driver.

This method is the same as the C function OGRGetDriverByName

Parameters

pszName	the driver name

Returns

the driver, or NULL if no driver with that name is found

4.6.2.4 int OGRSFDriverRegistrar::GetDriverCount (void)

Fetch the number of registered drivers.

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

Returns

the drivers count.

4.6.2.5 OGRDataSource * OGRSFDriverRegistrar::GetOpenDS (int *iDS*)

Return the iDS th datasource opened.

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

Parameters

iDS	the index of the dataset to return (between 0 and GetOpenDSCount() - 1)

4.6.2.6 int OGRSFDriverRegistrar::GetOpenDSCount ( ) [inline]

Return the number of opened datasources.

This method is the same as the C function OGRGetOpenDSCount()

Returns

the number of opened datasources.

4.6.2.7 OGRSFDriverRegistrar * OGRSFDriverRegistrar::GetRegistrar ( ) [static]

Return the driver manager, creating one if none exist.

Returns

the driver manager.

4.6.2.8 OGRDataSource * OGRSFDriverRegistrar::Open (const char * *pszName*, int *bUpdate* = FALSE, OGRSFDriver ** *ppoDriver* = NULL) [static]

Open a file / data source with one of the registered drivers.

This method loops through all the drivers registered with the driver manager trying each until one succeeds with the given data source. This method is static. Applications don't normally need to use any other OGRSFDriverRegistrar methods directly, nor do they normally need to have a pointer to an OGRSFDriverRegistrar instance.

If this method fails, CPLGetLastErrorMsg() can be used to check if there is an error message explaining why.

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

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

Parameters

pszName	the name of the file, or data source to open. UTF-8 encoded.
bUpdate	FALSE for read-only access (the default) or TRUE for read-write access.
ppoDriver	if non-NULL, this argument will be updated with a pointer to the driver which was used to open the data source.

Returns

NULL on error or if the pass name is not supported by this driver, otherwise a pointer to an OGRDataSource. This OGRDataSource should be closed by deleting the object when it is no longer needed.

Example:

```
OGRDataSource *poDS;

poDS = OGRSFDriverRegistrar::Open( "polygon.shp" );
if( poDS == NULL )
{
    return;
}

... use the data source ...

OGRDataSource::DestroyDataSource(poDS);
```

4.6.2.9 void OGRSFDriverRegistrar::RegisterDriver (OGRSFDriver * *poDriver*)

Add a driver to the list of registered drivers.

If the passed driver is already registered (based on pointer comparison) then the driver isn't registered. New drivers are added at the end of the list of registered drivers.

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

Parameters

poDriver	the driver to add.

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

- ogrsf_frmts.h
- ogrsf_frmts.dox

Chapter 5

File Documentation

5.1 ogrsf_frmts.h File Reference

```
#include "cpl_progress.h"
#include "ogr_feature.h"
#include "ogr_featurestyle.h"
```

Classes

- class OGRLayer
- class OGRDataSource
- class OGRSFDriver
- class OGRSFDriverRegistrar

Functions

- CPL_C_START void CPL_DLL OGRRegisterAll ()

  Register all drivers.
- void CPL_DLL **RegisterOGRFileGDB** ()
- void CPL_DLL **RegisterOGRShape** ()
- void CPL_DLL **RegisterOGRNTF** ()
- void CPL_DLL **RegisterOGRFME** ()
- void CPL_DLL **RegisterOGRSDTS** ()
- void CPL_DLL **RegisterOGRTiger** ()
- void CPL_DLL **RegisterOGRS57** ()
- void CPL_DLL **RegisterOGRTAB** ()
- void CPL_DLL **RegisterOGRMIF** ()
- void CPL_DLL **RegisterOGROGDI** ()
- void CPL_DLL **RegisterOGRODBC** ()
- void CPL_DLL **RegisterOGRWAsP** ()
- void CPL_DLL **RegisterOGRPG** ()
- void CPL_DLL **RegisterOGRMSSQLSpatial** ()
- void CPL_DLL **RegisterOGRMySQL** ()
- void CPL_DLL **RegisterOGROCI** ()
- void CPL_DLL **RegisterOGRDGN** ()
- void CPL_DLL **RegisterOGRGML** ()
- void CPL_DLL **RegisterOGRLIBKML** ()
- void CPL_DLL **RegisterOGRKML** ()

- void CPL_DLL **RegisterOGRGeoJSON** ()
- void CPL_DLL **RegisterOGRAVCBin** ()
- void CPL_DLL **RegisterOGRAVCE00** ()
- void CPL_DLL **RegisterOGRREC** ()
- void CPL_DLL **RegisterOGRMEM** ()
- void CPL_DLL **RegisterOGRVRT** ()
- void CPL_DLL **RegisterOGRDODS** ()
- void CPL_DLL **RegisterOGRSQLite** ()
- void CPL_DLL **RegisterOGRCSV** ()
- void CPL_DLL **RegisterOGRILI1** ()
- void CPL_DLL **RegisterOGRILI2** ()
- void CPL_DLL **RegisterOGRGRASS** ()
- void CPL_DLL **RegisterOGRPGeo** ()
- void CPL_DLL **RegisterOGRDXFDWG** ()
- void CPL_DLL **RegisterOGRDXF** ()
- void CPL_DLL **RegisterOGRDWG** ()
- void CPL_DLL **RegisterOGRSDE** ()
- void CPL_DLL **RegisterOGRIDB** ()
- void CPL_DLL **RegisterOGRGMT** ()
- void CPL_DLL **RegisterOGRBNA** ()
- void CPL_DLL **RegisterOGRGPX** ()
- void CPL_DLL **RegisterOGRGeoconcept** ()
- void CPL_DLL **RegisterOGRIngres** ()
- void CPL_DLL **RegisterOGRPCIDSK** ()
- void CPL_DLL **RegisterOGRXPlane** ()
- void CPL_DLL **RegisterOGRNAS** ()
- void CPL_DLL **RegisterOGRGeoRSS** ()
- void CPL_DLL **RegisterOGRGTM** ()
- void CPL_DLL **RegisterOGRVFK** ()
- void CPL_DLL **RegisterOGRPGDump** ()
- void CPL_DLL **RegisterOGROSM** ()
- void CPL_DLL **RegisterOGRGPSBabel** ()
- void CPL_DLL **RegisterOGRSUA** ()
- void CPL_DLL **RegisterOGROpenAir** ()
- void CPL_DLL **RegisterOGRPDS** ()
- void CPL_DLL **RegisterOGRWFS** ()
- void CPL_DLL **RegisterOGRSOSI** ()
- void CPL_DLL **RegisterOGRHTF** ()
- void CPL_DLL **RegisterOGRAeronavFAA** ()
- void CPL_DLL **RegisterOGRGeomedia** ()
- void CPL_DLL **RegisterOGRMDB** ()
- void CPL_DLL **RegisterOGREDIGEO** ()
- void CPL_DLL **RegisterOGRGFT** ()
- void CPL_DLL **RegisterOGRGME** ()
- void CPL_DLL **RegisterOGRSVG** ()
- void CPL_DLL **RegisterOGRCouchDB** ()
- void CPL_DLL **RegisterOGRIdrisi** ()
- void CPL_DLL **RegisterOGRARCGEN** ()
- void CPL_DLL **RegisterOGRSEGUKOOA** ()
- void CPL_DLL **RegisterOGRSEGY** ()
- void CPL_DLL **RegisterOGRXLS** ()
- void CPL_DLL **RegisterOGRODS** ()
- void CPL_DLL **RegisterOGRXLSX** ()
- void CPL_DLL **RegisterOGRElastic** ()
- void CPL_DLL **RegisterOGRGeoPackage** ()

- void CPL_DLL **RegisterOGRPDF** ()
- void CPL_DLL **RegisterOGRWalk** ()
- void CPL_DLL **RegisterOGRCartoDB** ()
- void CPL_DLL **RegisterOGRSXF** ()
- void CPL_DLL **RegisterOGROpenFileGDB** ()

5.1.1 Detailed Description

Classes related to registration of format support, and opening datasets.