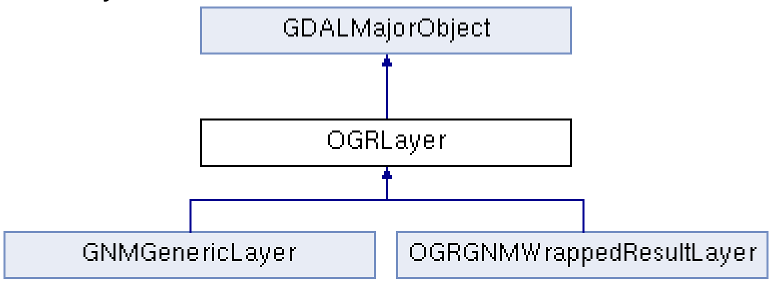
OGRLayer Class Reference abstract

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

```
#include <ogrsf_frmts.h>
```

Inheritance diagram for OGRLayer:



Public Member Functions

FeatureIterator	**begin** () Return begin of feature iterator. More...
FeatureIterator	**end** () Return end of feature iterator. More...
virtual **OGRGeometry** *	**GetSpatialFilter** () This method returns the current spatial filter for this layer. More...
virtual void	**SetSpatialFilter** (**OGRGeometry** *) Set a new spatial filter. More...
virtual void	**SetSpatialFilterRect** (double dfMinX, double dfMinY, double dfMaxX, double dfMaxY) Set a new rectangular spatial filter. More...
virtual void	**SetSpatialFilter** (int iGeomField, **OGRGeometry** *) Set a new spatial filter. More...
virtual void	**SetSpatialFilterRect** (int iGeomField, double dfMinX, double dfMinY, double dfMaxX, double dfMaxY) Set a new rectangular spatial filter. More...
virtual **OGRErr**	**SetAttributeFilter** (const char *) Set a new attribute query. More...
virtual void	**ResetReading** ()=0 Reset feature reading to start on the first feature. More...
virtual **OGRFeature** *	**GetNextFeature** () **CPL_WARN_UNUSED_RESULT**=0 Fetch the next available feature from this layer. More...
virtual **OGRErr**	**SetNextByIndex** (**GIntBig** nIndex) Move read cursor to the nIndex'th feature in the current resultset. More...
virtual **OGRFeature** *	**GetFeature** (**GIntBig** nFID) **CPL_WARN_UNUSED_RESULT** Fetch a feature by its identifier. More...

OGRErr	**SetFeature** (**OGRFeature** *poFeature) **CPL_WARN_UNUSED_RESULT** Rewrite an existing feature. More...
OGRErr	**CreateFeature** (**OGRFeature** *poFeature) **CPL_WARN_UNUSED_RESULT** Create and write a new feature within a layer. More...
virtual OGRErr	**DeleteFeature** (**GIntBig** nFID) **CPL_WARN_UNUSED_RESULT** Delete feature from layer. More...
virtual const char *	**GetName** () Return the layer name. More...
virtual OGRwkbGeometryType	**GetGeomType** () Return the layer geometry type. More...
virtual OGRFeatureDefn *	**GetLayerDefn** ()=0 Fetch the schema information for this layer. More...
virtual int	**FindFieldIndex** (const char *pszFieldName, int bExactMatch) Find the index of field in the layer. More...
virtual OGRSpatialReference *	**GetSpatialRef** () Fetch the spatial reference system for this layer. More...
virtual GIntBig	**GetFeatureCount** (int bForce=TRUE) Fetch the feature count in this layer. More...
virtual OGRErr	**GetExtent** (OGREnvelope *psExtent, int bForce=TRUE) **CPL_WARN_UNUSED_RESULT** Fetch the extent of this layer. More...
virtual OGRErr	**GetExtent** (int iGeomField, OGREnvelope *psExtent, int bForce=TRUE) **CPL_WARN_UNUSED_RESULT** Fetch the extent of this layer, on the specified geometry field. More...
virtual int	**TestCapability** (const char *)=0 Test if this layer supported the named capability. More...
virtual OGRErr	**CreateField** (**OGRFieldDefn** *poField, int bApproxOK=TRUE) Create a new field on a layer. More...
virtual OGRErr	**DeleteField** (int iField) Delete an existing field on a layer. More...
virtual OGRErr	**ReorderFields** (int *panMap) Reorder all the fields of a layer. More...
virtual OGRErr	**AlterFieldDefn** (int iField, **OGRFieldDefn** *poNewFieldDefn, int nFlagsIn) Alter the definition of an existing field on a layer. More...
virtual OGRErr	**CreateGeomField** (**OGRGeomFieldDefn** *poField, int bApproxOK=TRUE) Create a new geometry field on a layer. More...
virtual OGRErr	**SyncToDisk** () Flush pending changes to disk. More...

virtual **OGRStyleTable** *	**GetStyleTable** () Returns layer style table. More...
virtual void	**SetStyleTableDirectly** (**OGRStyleTable** *poStyleTable) Set layer style table. More...
virtual void	**SetStyleTable** (**OGRStyleTable** *poStyleTable) Set layer style table. More...
virtual **OGRErr**	**StartTransaction** () **CPL_WARN_UNUSED_RESULT** For datasources which support transactions, StartTransaction creates a transaction. More...
virtual **OGRErr**	**CommitTransaction** () **CPL_WARN_UNUSED_RESULT** 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...
virtual const char *	**GetFIDColumn** () This method returns the name of the underlying database column being used as the FID column, or "" if not supported. More...
virtual const char *	**GetGeometryColumn** () This method returns the name of the underlying database column being used as the geometry column, or "" if not supported. More...
virtual **OGRErr**	**SetIgnoredFields** (const char **papszFields) Set which fields can be omitted when retrieving features from the layer. More...
OGRErr	**Intersection** (**OGRLayer** *pLayerMethod, **OGRLayer** *pLayerResult, char **papszOptions=nullptr, GDALProgressFunc pfnProgress=nullptr, void *pProgressArg=nullptr) Intersection of two layers. More...
OGRErr	**Union** (**OGRLayer** *pLayerMethod, **OGRLayer** *pLayerResult, char **papszOptions=nullptr, GDALProgressFunc pfnProgress=nullptr, void *pProgressArg=nullptr) Union of two layers. More...
OGRErr	**SymDifference** (**OGRLayer** *pLayerMethod, **OGRLayer** *pLayerResult, char **papszOptions, GDALProgressFunc pfnProgress, void *pProgressArg) Symmetrical difference of two layers. More...
OGRErr	**Identity** (**OGRLayer** *pLayerMethod, **OGRLayer** *pLayerResult, char **papszOptions=nullptr, GDALProgressFunc pfnProgress=nullptr, void *pProgressArg=nullptr) Identify the features of this layer with the ones from the identity layer. More...
OGRErr	**Update** (**OGRLayer** *pLayerMethod, **OGRLayer** *pLayerResult, char

	**papszOptions=nullptr, GDALProgressFunc pfnProgress=nullptr, void *pProgressArg=nullptr) Update this layer with features from the update layer. More...
OGRErr	**Clip** (**OGRLayer** *pLayerMethod, **OGRLayer** *pLayerResult, char **papszOptions=nullptr, GDALProgressFunc pfnProgress=nullptr, void *pProgressArg=nullptr) Clip off areas that are not covered by the method layer. More...
OGRErr	**Erase** (**OGRLayer** *pLayerMethod, **OGRLayer** *pLayerResult, char **papszOptions=nullptr, GDALProgressFunc pfnProgress=nullptr, void *pProgressArg=nullptr) Remove areas that are covered by the method layer. More...
int	**Reference** () Increment layer reference count. More...
int	**Dereference** () Decrement layer reference count. More...
int	**GetRefCount** () const Fetch reference count. More...
OGRErr	**ReorderField** (int iOldFieldPos, int iNewFieldPos) Reorder an existing field on a layer. More...

▸ **Public Member Functions inherited from GDALMajorObject**

Static Public Member Functions

static **OGRLayerH**	**ToHandle** (**OGRLayer** *poLayer) Convert a OGRLayer* to a OGRLayerH. More...
static **OGRLayer** *	**FromHandle** (**OGRLayerH** hLayer) Convert a OGRLayerH to a OGRLayer*. More...

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

Protected Member Functions

virtual **OGRErr**	**ISetFeature** (**OGRFeature** *poFeature) **CPL_WARN_UNUSED_RESULT** Rewrite an existing feature. More...
virtual **OGRErr**	**ICreateFeature** (**OGRFeature** *poFeature) **CPL_WARN_UNUSED_RESULT** Create and write a new feature within a layer. More...

▸ **Protected Member Functions inherited from GDALMajorObject**

Friends

FeatureIterator	**begin** (**OGRLayer** *poLayer) Return begin of feature iterator. More...
FeatureIterator	**end** (**OGRLayer** *poLayer) Return end of feature iterator. More...

Detailed Description

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

Member Function Documentation

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

```
OGRErr OGRLayer::AlterFieldDefn ( int            iField,
                                  OGRFieldDefn * poNewFieldDefn,
                                  int            nFlags
                                )
```

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 existence 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 existing 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, ALTER_WIDTH_PRECISION_FLAG, ALTER_NULLABLE_FLAG and ALTER_DEFAULT_FLAG to indicate which of the name and/or type and/or width and precision fields and/or nullability from the new field definition must be taken into account.

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

Reimplemented in **GNMGenericLayer**.

OGRLayer::FeatureIterator OGRLayer::begin ( )

Return begin of feature iterator.

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 OGRFeatureUniquePtr reference is reused.

Only one iterator per layer can be active at a time.

Since

GDAL 2.3

```
OGRErr OGRLayer::Clip ( OGRLayer *          pLayerMethod,
                        OGRLayer *          pLayerResult,
                        char **             papszOptions = nullptr,
                        GDALProgressFunc    pfnProgress = nullptr,
                        void *              pProgressArg = nullptr
                      )
```

Clip off areas that are not covered by the method layer.

The result layer contains features whose geometries represent areas that are in the input layer and in the method layer. The features in the result layer have the (possibly clipped) areas of features in the input layer and the attributes from the same features. The schema of the result layer can be set by the user or, if it is empty, is initialized to contain all fields in the input layer.

Note

For best performance use the minimum amount of features in the method layer and copy it into a memory layer.

This method relies on GEOS support. Do not use unless the GEOS support is compiled in.

The recognized list of options is :

- SKIP_FAILURES=YES/NO. Set it to YES to go on, even when a feature could not be inserted or a GEOS call failed.
- PROMOTE_TO_MULTI=YES/NO. Set it to YES to convert Polygons into MultiPolygons, or LineStrings to MultiLineStrings.
- INPUT_PREFIX=string. Set a prefix for the field names that will be created from the fields of the input layer.
- METHOD_PREFIX=string. Set a prefix for the field names that will be created from the fields of the method layer.

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

Parameters

pLayerMethod	the method layer. Should not be NULL.
pLayerResult	the layer where the features resulting from the operation are inserted. Should not be NULL. See above the note about the schema.
papszOptions	NULL terminated list of options (may be NULL).
pfnProgress	a GDALProgressFunc() compatible callback function for reporting progress or NULL.
pProgressArg	argument to be passed to pfnProgress. May be NULL.

Returns

an error code if there was an error or the execution was interrupted, OGRERR_NONE otherwise.

Note

The first geometry field is always used.

Since

OGR 1.10

OGRErr OGRLayer::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_NONE.

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

Returns

OGRERR_NONE on success.

Reimplemented in **GNMGenericLayer**.

OGRErr OGRLayer::CreateFeature (OGRFeature * poFeature)

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.

Starting with GDAL 2.0, drivers should specialize the **ICreateFeature()** method, since **CreateFeature()** is no longer virtual.

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.

```
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 existence 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 existing features of the backing file/database should be updated accordingly.

Drivers may or may not support not-null constraints. If they support creating fields with not-null constraints, this is generally before creating any feature to the layer.

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.

Reimplemented in **OGRGNMWrappedResultLayer**, and **GNMGenericLayer**.

```
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 existence 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 existing features of the backing file/database should be updated accordingly.

Drivers may or may not support not-null constraints. If they support creating fields with not-null constraints, this is generally before creating any feature to the layer.

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

Reimplemented in **OGRGNMWrappedResultLayer**, and **GNMGenericLayer**.

OGRErr OGRLayer::DeleteFeature (GIntBig 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 if the operation works, otherwise an appropriate error code (e.g OGRERR_NON_EXISTING_FEATURE if the feature does not exist).

Reimplemented in **GNMGenericLayer**.

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 existence 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 existing 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

Reimplemented in **GNMGenericLayer**.

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.

OGRLayer::FeatureIterator OGRLayer::end ( )

Return end of feature iterator.

```
OGRErr OGRLayer::Erase ( OGRLayer *          pLayerMethod,
                         OGRLayer *          pLayerResult,
                         char **             papszOptions = nullptr,
                         GDALProgressFunc    pfnProgress = nullptr,
                         void *              pProgressArg = nullptr
                       )
```

Remove areas that are covered by the method layer.

The result layer contains features whose geometries represent areas that are in the input layer but not in the method layer. The features in the result layer have attributes from the input layer. The schema of the result layer can be set by the user or, if it is empty, is initialized to contain all fields in the input layer.

Note

For best performance use the minimum amount of features in the method layer and copy it into a memory layer.

This method relies on GEOS support. Do not use unless the GEOS support is compiled in.

The recognized list of options is :

- SKIP_FAILURES=YES/NO. Set it to YES to go on, even when a feature could not be inserted or a GEOS call failed.
- PROMOTE_TO_MULTI=YES/NO. Set it to YES to convert Polygons into MultiPolygons, or LineStrings to MultiLineStrings.
- INPUT_PREFIX=string. Set a prefix for the field names that will be created from the fields of the input layer.
- METHOD_PREFIX=string. Set a prefix for the field names that will be created from the fields of the method layer.

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

Parameters

pLayerMethod	the method layer. Should not be NULL.
pLayerResult	the layer where the features resulting from the operation are inserted. Should not be NULL. See above the note about the schema.
papszOptions	NULL terminated list of options (may be NULL).
pfnProgress	a GDALProgressFunc() compatible callback function for reporting progress or NULL.
pProgressArg	argument to be passed to pfnProgress. May be NULL.

Returns

an error code if there was an error or the execution was interrupted, OGRERR_NONE otherwise.

Note

The first geometry field is always used.

Since

OGR 1.10

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

Reimplemented in **GNMGenericLayer**.

static OGRLayer* OGRLayer::FromHandle (OGRLayerH hLayer) inline static

Convert a OGRLayerH to a OGRLayer*.

Since

GDAL 2.3

```
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.

Reimplemented in **GNMGenericLayer**.

```
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 implementer: 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.

Reimplemented in **GNMGenericLayer**.

OGRFeature * OGRLayer::GetFeature (GIntBig 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.

Reimplemented in **OGRGNMWrappedResultLayer**.

GIntBig 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()**.

Note: since GDAL 2.0, this method returns a GIntBig (previously a int)

Parameters

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

Returns

feature count, -1 if count not known.

Reimplemented in **OGRGNMWrappedResultLayer**, and **GNMGenericLayer**.

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.

Reimplemented in **OGRGNMWrappedResultLayer**, and **GNMGenericLayer**.

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.

For layers with multiple geometry fields, this method only returns the name of the first geometry column. For other columns, use GetLayerDefn()->OGRFeatureDefn::GetGeomFieldDefn(i)->GetNameRef().

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

Returns

geometry column name.

Reimplemented in OGRGNMWrappedResultLayer, and GNMGenericLayer.

OGRwkbGeometryType OGRLayer::GetGeomType ( ) virtual

Return the layer geometry type.

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

For layers with multiple geometry fields, this method only returns the geometry type of the first geometry column. For other columns, use GetLayerDefn()->OGRFeatureDefn::GetGeomFieldDefn(i)->GetType(). For layers without any geometry field, this method returns wkbNone.

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()->OGRFeatureDefn::GetGeomType().

Returns

the geometry type

Since

OGR 1.8.0

Reimplemented in GNMGenericLayer.

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.

Implemented in OGRGNMWrappedResultLayer, and GNMGenericLayer.

const char * OGRLayer::GetName ( ) virtual

Return the layer name.

This returns the same content as GetLayerDefn()->OGRFeatureDefn::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()->OGRFeatureDefn::GetName().

Returns

the layer name (must not been freed)

Since

OGR 1.8.0

Reimplemented in GNMGenericLayer.

OGRFeature * OGRLayer::GetNextFeature ( ) pure virtual

Fetch the next available feature from this layer.

The returned feature becomes the responsibility 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.

Features returned by GetNextFeature() may or may not be affected by concurrent modifications depending on drivers. A guaranteed way of seeing modifications in effect is to call ResetReading() on layers where GetNextFeature() has been called, before reading again. Structural changes in layers (field addition, deletion, ...) when a read is in progress may or may not be possible depending on drivers. If a transaction is committed/aborted, the current sequential reading may or may not be valid after that operation and a call to ResetReading() might be needed.

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

Returns

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

Implemented in OGRGNMWrappedResultLayer, and GNMGenericLayer.

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.

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.

Reimplemented in **GNMGenericLayer**.

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()**->OGRFeatureDefn::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.

Reimplemented in **OGRGNMWrappedResultLayer**, and **GNMGenericLayer**.

OGRStyleTable * 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.

Reimplemented in **GNMGenericLayer**.

OGRErr OGRLayer::ICreateFeature (OGRFeature * poFeature) protected virtual

Create and write a new feature within a layer.

This method is implemented by drivers and not called directly. User code should use CreateFeature() instead.

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.

Parameters

poFeature	the feature to write to disk.

Returns

OGRERR_NONE on success.

Since

GDAL 2.0

Reimplemented in OGRGNMWrappedResultLayer.

```
OGRErr OGRLayer::Identity ( OGRLayer *          pLayerMethod,
                            OGRLayer *          pLayerResult,
                            char **             papszOptions = nullptr,
                            GDALProgressFunc    pfnProgress = nullptr,
                            void *              pProgressArg = nullptr
                          )
```

Identify the features of this layer with the ones from the identity layer.

The result layer contains features whose geometries represent areas that are in the input layer. The features in the result layer have attributes from both input and method layers. The schema of the result layer can be set by the user or, if it is empty, is initialized to contain all fields in input and method layers.

Note

If the schema of the result is set by user and contains fields that have the same name as a field in input and in method layer, then the attribute in the result feature will get the value from the feature of the method layer (even if it is undefined).

For best performance use the minimum amount of features in the method layer and copy it into a memory layer.

This method relies on GEOS support. Do not use unless the GEOS support is compiled in.

The recognized list of options is :

- SKIP_FAILURES=YES/NO. Set it to YES to go on, even when a feature could not be inserted or a GEOS call failed.
- PROMOTE_TO_MULTI=YES/NO. Set it to YES to convert Polygons into MultiPolygons, or LineStrings to MultiLineStrings.
- INPUT_PREFIX=string. Set a prefix for the field names that will be created from the fields of the input layer.
- METHOD_PREFIX=string. Set a prefix for the field names that will be created from the fields of the method layer.
- USE_PREPARED_GEOMETRIES=YES/NO. Set to NO to not use prepared geometries to pretest intersection of features of method layer with features of this layer.
- KEEP_LOWER_DIMENSION_GEOMETRIES=YES/NO. Set to NO to skip result features with lower dimension geometry that would otherwise be added to the result layer. The default is to add but only if the result layer has an unknown geometry type.

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

Parameters

pLayerMethod	the method layer. Should not be NULL.
pLayerResult	the layer where the features resulting from the operation are inserted. Should not be NULL. See above the note about the schema.
papszOptions	NULL terminated list of options (may be NULL).
pfnProgress	a GDALProgressFunc() compatible callback function for reporting progress or

NULL.

pProgressArg	argument to be passed to pfnProgress. May be NULL.

Returns

an error code if there was an error or the execution was interrupted, OGRERR_NONE otherwise.

Note

The first geometry field is always used.

Since

OGR 1.10

```
OGRErr OGRLayer::Intersection ( OGRLayer *         pLayerMethod,
                                OGRLayer *         pLayerResult,
                                char **            papszOptions = nullptr,
                                GDALProgressFunc   pfnProgress = nullptr,
                                void *             pProgressArg = nullptr
                              )
```

Intersection of two layers.

The result layer contains features whose geometries represent areas that are common between features in the input layer and in the method layer. The features in the result layer have attributes from both input and method layers. The schema of the result layer can be set by the user or, if it is empty, is initialized to contain all fields in the input and method layers.

Note

If the schema of the result is set by user and contains fields that have the same name as a field in input and in method layer, then the attribute in the result feature will get the value from the feature of the method layer.

For best performance use the minimum amount of features in the method layer and copy it into a memory layer.

This method relies on GEOS support. Do not use unless the GEOS support is compiled in.

The recognized list of options is:

- SKIP_FAILURES=YES/NO. Set to YES to go on, even when a feature could not be inserted or a GEOS call failed.
- PROMOTE_TO_MULTI=YES/NO. Set to YES to convert Polygons into MultiPolygons, or LineStrings to MultiLineStrings.
- INPUT_PREFIX=string. Set a prefix for the field names that will be created from the fields of the input layer.
- METHOD_PREFIX=string. Set a prefix for the field names that will be created from the fields of the method layer.
- USE_PREPARED_GEOMETRIES=YES/NO. Set to NO to not use prepared geometries to pretest intersection of features of method layer with features of this layer.
- PRETEST_CONTAINMENT=YES/NO. Set to YES to pretest the containment of features of method layer within the features of this layer. This will speed up the method significantly in some cases. Requires that the prepared geometries are in effect.
- KEEP_LOWER_DIMENSION_GEOMETRIES=YES/NO. Set to NO to skip result features with lower dimension geometry that would otherwise be added to the result layer. The default is to add but only if the result layer has an unknown geometry type.

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

Parameters

pLayerMethod	the method layer. Should not be NULL.
pLayerResult	the layer where the features resulting from the operation are inserted. Should not be NULL. See above the note about the schema.

papszOptions	NULL terminated list of options (may be NULL).
pfnProgress	a GDALProgressFunc() compatible callback function for reporting progress or NULL.
pProgressArg	argument to be passed to pfnProgress. May be NULL.

Returns

an error code if there was an error or the execution was interrupted, OGRERR_NONE otherwise.

Note

The first geometry field is always used.

Since

OGR 1.10

OGRErr OGRLayer::ISetFeature (OGRFeature * **poFeature**) protected virtual

Rewrite an existing feature.

This method is implemented by drivers and not called directly. User code should use **SetFeature()** instead.

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

Parameters

poFeature the feature to write.

Returns

OGRERR_NONE if the operation works, otherwise an appropriate error code (e.g OGRERR_NON_EXISTING_FEATURE if the feature does not exist).

Since

GDAL 2.0

Reimplemented in **OGRGNMWrappedResultLayer**.

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.

```
OGRErr OGRLayer::ReorderField ( int  iOldFieldPos,
                                int  iNewFieldPos
                              )
```

Reorder an existing field on a layer.

This method is a convenience 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 existence 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 existing 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

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 existence 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 existing 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()**->**OGRFeatureDefn::GetFieldCount()** elements which is a permutation of [0, **GetLayerDefn()**->**OGRFeatureDefn::GetFieldCount()**-1].

Returns

OGRERR_NONE on success.

Since

OGR 1.9.0

Reimplemented in **GNMGenericLayer**.

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()**.

Implemented in **OGRGNMWrappedResultLayer**, and **GNMGenericLayer**.

OGRErr OGRLayer::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_NONE.

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

Returns

OGRERR_NONE on success.

Reimplemented in **GNMGenericLayer**.

OGRErr 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 interpret 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.

Reimplemented in **GNMGenericLayer**.

OGRErr OGRLayer::SetFeature (OGRFeature * poFeature)

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()**.

Starting with GDAL 2.0, drivers should specialize the **ISetFeature()** method, since **SetFeature()** is no longer virtual.

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 (e.g OGRERR_NON_EXISTING_FEATURE if the feature does not exist).

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)

Reimplemented in **GNMGenericLayer**.

OGRErr OGRLayer::SetNextByIndex (GIntBig 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.

Reimplemented in **OGRGNMWrappedResultLayer**, and **GNMGenericLayer**.

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 whose 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.

Starting with GDAL 2.3, features with null or empty geometries will never be considered as matching a spatial filter.

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.

Reimplemented in **GNMGenericLayer**.

```
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()**->OGRFeatureDefn::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 implementer: 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

Reimplemented in **GNMGenericLayer**.

```
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.

Reimplemented in **GNMGenericLayer**.

```
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()**->OGRFeatureDefn::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

Reimplemented in **GNMGenericLayer**.

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

Reimplemented in GNMGenericLayer.

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

Reimplemented in GNMGenericLayer.

OGRErr OGRLayer::StartTransaction ( ) 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_NONE.

Note: as of GDAL 2.0, use of this API is discouraged when the dataset offers dataset level transaction with GDALDataset::StartTransaction(). The reason is that most drivers can only offer transactions at dataset level, and not layer level. Very few drivers really support transactions at layer scope.

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

Returns

OGRERR_NONE on success.

Reimplemented in GNMGenericLayer.

```
OGRErr OGRLayer::SymDifference ( OGRLayer *          pLayerMethod,
                                 OGRLayer *          pLayerResult,
                                 char **             papszOptions,
                                 GDALProgressFunc    pfnProgress,
                                 void *              pProgressArg
                               )
```

Symmetrical difference of two layers.

The result layer contains features whose geometries represent areas that are in either in the input layer or in the method layer but not in both. The features in the result layer have attributes from both input and method layers. For features which represent areas that are only in the input or in the method layer the respective attributes have undefined values. The schema of the result layer can be set by the user or, if it is empty, is initialized to contain all fields in the input and method layers.

Note

If the schema of the result is set by user and contains fields that have the same name as a field in input and in method layer, then the attribute in the result feature will get the value from the feature of the method layer (even if it is undefined).

For best performance use the minimum amount of features in the method layer and copy it into a memory layer.

This method relies on GEOS support. Do not use unless the GEOS support is compiled in.

The recognized list of options is :

- SKIP_FAILURES=YES/NO. Set it to YES to go on, even when a feature could not be inserted or a GEOS call failed.
- PROMOTE_TO_MULTI=YES/NO. Set it to YES to convert Polygons into MultiPolygons, or LineStrings to MultiLineStrings.
- INPUT_PREFIX=string. Set a prefix for the field names that will be created from the fields of the input layer.
- METHOD_PREFIX=string. Set a prefix for the field names that will be created from the fields of the method layer.

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

Parameters

pLayerMethod	the method layer. Should not be NULL.
pLayerResult	the layer where the features resulting from the operation are inserted. Should not be NULL. See above the note about the schema.
papszOptions	NULL terminated list of options (may be NULL).
pfnProgress	a GDALProgressFunc() compatible callback function for reporting progress or NULL.
pProgressArg	argument to be passed to pfnProgress. May be NULL.

Returns

an error code if there was an error or the execution was interrupted, OGRERR_NONE otherwise.

Note

The first geometry field is always used.

Since

OGR 1.10

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.

Reimplemented in **GNMGenericLayer**.

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. i.e. 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, i.e. 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.
- **OLCCurveGeometries** / "CurveGeometries": TRUE if this layer supports writing curve geometries or may return such geometries. (GDAL 2.0).

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 unrecognized capabilities.

Implemented in **OGRGNMWrappedResultLayer**, and **GNMGenericLayer**.

static OGRLayerH OGRLayer::ToHandle (OGRLayer * poLayer) inline static

Convert a OGRLayer* to a OGRLayerH.

Since

GDAL 2.3

```
OGRErr OGRLayer::Union ( OGRLayer *          pLayerMethod,
                         OGRLayer *          pLayerResult,
                         char **             papszOptions = nullptr,
                         GDALProgressFunc    pfnProgress = nullptr,
                         void *              pProgressArg = nullptr
                       )
```

Union of two layers.

The result layer contains features whose geometries represent areas that are either in the input layer, in the method layer, or in both. The features in the result layer have attributes from both input and method layers. For features which represent areas that are only in the input or in the method layer the respective attributes have undefined values. The schema of the result layer can be set by the user or, if it is empty, is initialized to contain all fields in the input and method layers.

Note

If the schema of the result is set by user and contains fields that have the same name as a field in input and in method layer, then the attribute in the result feature will get the value from the feature of the method layer (even if it is undefined).

For best performance use the minimum amount of features in the method layer and copy it into a memory layer.

This method relies on GEOS support. Do not use unless the GEOS support is compiled in.

The recognized list of options is :

- SKIP_FAILURES=YES/NO. Set it to YES to go on, even when a feature could not be inserted or a GEOS call failed.
- PROMOTE_TO_MULTI=YES/NO. Set it to YES to convert Polygons into MultiPolygons, or LineStrings to MultiLineStrings.
- INPUT_PREFIX=string. Set a prefix for the field names that will be created from the fields of the input layer.
- METHOD_PREFIX=string. Set a prefix for the field names that will be created from the fields of the method layer.
- USE_PREPARED_GEOMETRIES=YES/NO. Set to NO to not use prepared geometries to pretest intersection of features of method layer with features of this layer.
- KEEP_LOWER_DIMENSION_GEOMETRIES=YES/NO. Set to NO to skip result features with lower dimension geometry that would otherwise be added to the result layer. The default is to add but only if the result layer has an unknown geometry type.

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

Parameters

pLayerMethod	the method layer. Should not be NULL.
pLayerResult	the layer where the features resulting from the operation are inserted. Should not be NULL. See above the note about the schema.
papszOptions	NULL terminated list of options (may be NULL).

pfnProgress	a GDALProgressFunc() compatible callback function for reporting progress or NULL.
pProgressArg	argument to be passed to pfnProgress. May be NULL.

Returns

an error code if there was an error or the execution was interrupted, OGRERR_NONE otherwise.

Note

The first geometry field is always used.

Since

OGR 1.10

```
OGRErr OGRLayer::Update ( OGRLayer *          pLayerMethod,
                          OGRLayer *          pLayerResult,
                          char **             papszOptions = nullptr,
                          GDALProgressFunc    pfnProgress = nullptr,
                          void *              pProgressArg = nullptr
                        )
```

Update this layer with features from the update layer.

The result layer contains features whose geometries represent areas that are either in the input layer or in the method layer. The features in the result layer have areas of the features of the method layer or those ares of the features of the input layer that are not covered by the method layer. The features of the result layer get their attributes from the input layer. The schema of the result layer can be set by the user or, if it is empty, is initialized to contain all fields in the input layer.

Note

If the schema of the result is set by user and contains fields that have the same name as a field in the method layer, then the attribute in the result feature the originates from the method layer will get the value from the feature of the method layer.

For best performance use the minimum amount of features in the method layer and copy it into a memory layer.

This method relies on GEOS support. Do not use unless the GEOS support is compiled in.

The recognized list of options is :

- SKIP_FAILURES=YES/NO. Set it to YES to go on, even when a feature could not be inserted or a GEOS call failed.
- PROMOTE_TO_MULTI=YES/NO. Set it to YES to convert Polygons into MultiPolygons, or LineStrings to MultiLineStrings.
- INPUT_PREFIX=string. Set a prefix for the field names that will be created from the fields of the input layer.
- METHOD_PREFIX=string. Set a prefix for the field names that will be created from the fields of the method layer.

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

Parameters

pLayerMethod	the method layer. Should not be NULL.
pLayerResult	the layer where the features resulting from the operation are inserted. Should not be NULL. See above the note about the schema.
papszOptions	NULL terminated list of options (may be NULL).
pfnProgress	a GDALProgressFunc() compatible callback function for reporting progress or NULL.
pProgressArg	argument to be passed to pfnProgress. May be NULL.

Returns

an error code if there was an error or the execution was interrupted, OGRERR_NONE otherwise.

Note

The first geometry field is always used.

Since

OGR 1.10

Friends And Related Function Documentation

FeatureIterator begin (OGRLayer * poLayer) friend

Return begin of feature iterator.

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 std::unique_ptr<**OGRFeature**> reference is reused.

Only one iterator per layer can be active at a time.

Since

GDAL 2.3

See also

OGRLayer::begin()

FeatureIterator end (OGRLayer * poLayer) friend

Return end of feature iterator.

See also

OGRLayer::end()

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

- **ogrsf_frmts.h**
- ogrsf_frmts.dox
- ogrlayer.cpp