

OGR



# Contents



# Chapter 1

## OGR Simple Feature Library

The OGR Simple Features Library is a C++ open source library (and commandline tools) providing read (and sometimes write) access to a variety of vector file formats including ESRI Shapefiles, S-57, SDTS, PostGIS, Oracle Spatial, and Mapinfo mid/mif and TAB formats.

OGR is a part of the GDAL library.

### Resources

- OGR Supported Formats : ESRI Shapefile, ESRI ArcSDE, MapInfo (tab and mid/mif), GML, KML, PostGIS, Oracle Spatial, ...
- OGR Utility Programs : ogrinfo, ogr2ogr, ogrtindex
- OGR Class Documentation
- OGR C++ API Read/Write Tutorial
- OGR Driver Implementation Tutorial
- ogr\_api.h: OGR C API
- ogr\_srs\_api.h: OSR C API
- OGR Projections Tutorial
- OGR Architecture
- OGR SQL dialect and SQLITE SQL dialect
- OGR - Feature Style Specification
- Adam's 2.5 D Simple Features Proposal (OGC 99-402r2)
- Adam's SRS WKT Clarification Proposal in html or doc format.

### Download

#### Ready to Use Executables

The best way to get OGR utilities in ready-to-use form is to download the latest FWTools kit for your platform. While large, these include builds of the OGR utilities with lots of optional components built-in. Once downloaded follow the included instructions to setup your path and other environment variables correctly, and then you can use the various OGR utilities from the command line. The kits also include OpenEV, a viewer that will display OGR supported vector files.

## Source

The source code for this effort is intended to be available as OpenSource using an X Consortium style license. The OGR library is currently a loosely coupled subcomponent of the GDAL library, so you get all of GDAL for the "price" of OGR. See the [GDAL Download](#) and [Building](#) pages for details on getting the source and building it.

## Bug Reporting

GDAL/OGR bugs can be reported, and can be listed using [Trac](#).

## Mailing Lists

A `gdal-announce` mailing list subscription is a low volume way of keeping track of major developments with the GDAL/OGR project.

The `gdal-dev@lists.osgeo.org` mailing list can be used for discussion of development and user issues related to OGR and related technologies. Subscriptions can be done, and archives reviewed on the [web](#).

## Alternative Bindings for the OGR API

In addition to the C++ API primarily addressed in the online documentation, there is also a slightly less complete C API implemented on top of the C++ API, and access available from Python.

The C API is primarily intended to provide a less fragile API since slight changes in the C++ API (such as const correctness changes) can cause changes in method and class signatures that prevent use of new DLLs with older clients. The C API is also generally easy to call from other languages which allow call out to DLLs functions, such as Visual Basic, or Delphi. The API can be explored in the `ogr_api.h` include file. The `gdal/ogr/ogr_capi_test.c` is a small sample program demonstrating use of the C API.

The Python API isn't really well documented at this time, but parallels the C/C++ APIs. The interface classes can be browsed in the `pymod/ogr.py` (simple features) and `pymod/osr.py` (coordinate systems) python modules. The `pymod/samples/assemblepoly.py` sample script is one demonstration of using the python API.

## Chapter 2

# OGR API Tutorial

This document is intended to document using the OGR C++ classes to read and write data from a file. It is strongly advised that the read first review the [OGR Architecture](#) document describing the key classes and their roles in OGR.

It also includes code snippets for the corresponding functions in C and Python.

### 2.1 Reading From OGR

For purposes of demonstrating reading with OGR, we will construct a small utility for dumping point layers from an OGR data source to stdout in comma-delimited format.

Initially it is necessary to register all the format drivers that are desired. This is normally accomplished by calling **OGRRegisterAll()** (p. ??) which registers all format drivers built into GDAL/OGR.

In C++ :

```
#include "ogr_sfrmts.h"

int main()
{
    OGRRegisterAll();
}
```

In C :

```
#include "ogr_api.h"

int main()
{
    OGRRegisterAll();
}
```

Next we need to open the input OGR datasource. Datasources can be files, RDBMSes, directories full of files, or even remote web services depending on the driver being used. However, the datasource name is always a single string. In this case we are hardcoded to open a particular shapefile. The second argument (FALSE) tells the **OGRSFDriverRegistrar::Open()** (p. ??) method that we don't require update access. On failure NULL is returned, and we report an error.

In C++ :

```
OGRDataSource      *poDS;

poDS = OGRSFDriverRegistrar::Open( "point.shp", FALSE );
if( poDS == NULL )
{
    printf( "Open failed.\n" );
    exit( 1 );
}
```

In C :

```
OGRDataSourceH hDS;

hDS = OGROpen( "point.shp", FALSE, NULL );
if( hDS == NULL )
{
    printf( "Open failed.\n" );
    exit( 1 );
}
```

An **OGRDataSource** (p. ??) can potentially have many layers associated with it. The number of layers available can be queried with **OGRDataSource::GetLayerCount()** (p. ??) and individual layers fetched by index using **OGRDataSource::GetLayer()** (p. ??). However, we will just fetch the layer by name.

In C++ :

```
OGRLayer *poLayer;

poLayer = poDS->GetLayerByName( "point" );
```

In C :

```
OGRLayerH hLayer;

hLayer = OGR_DS_GetLayerByName( hDS, "point" );
```

Now we want to start reading features from the layer. Before we start we could assign an attribute or spatial filter to the layer to restrict the set of feature we get back, but for now we are interested in getting all features.

While it isn't strictly necessary in this circumstance since we are starting fresh with the layer, it is often wise to call **OGRLayer::ResetReading()** (p. ??) to ensure we are starting at the beginning of the layer. We iterate through all the features in the layer using **OGRLayer::GetNextFeature()** (p. ??). It will return NULL when we run out of features.

In C++ :

```
OGRFeature *poFeature;

poLayer->ResetReading();
while( (poFeature = poLayer->GetNextFeature()) != NULL )
{
```

In C :

```
OGRFeatureH hFeature;

OGR_L_ResetReading(hLayer);
while( (hFeature = OGR_L_GetNextFeature(hLayer)) != NULL )
{
```

In order to dump all the attribute fields of the feature, it is helpful to get the **OGRFeatureDefn** (p. ??). This is an object, associated with the layer, containing the definitions of all the fields. We loop over all the fields, and fetch and report the attributes based on their type.

In C++ :

```
OGRFeatureDefn *poFDefn = poLayer->GetLayerDefn();
int iField;

for( iField = 0; iField < poFDefn->GetFieldCount(); iField++ )
{
    OGRFieldDefn *poFieldDefn = poFDefn->GetFieldDefn( iField );

    if( poFieldDefn->GetType() == OFTInteger )
        printf( "%d,", poFeature->GetFieldAsInteger( iField ) );
    else if( poFieldDefn->GetType() == OFTReal )
        printf( "%.3f,", poFeature->GetFieldAsDouble(iField) );
    else if( poFieldDefn->GetType() == OFTString )
        printf( "%s,", poFeature->GetFieldAsString(iField) );
    else
        printf( "%s,", poFeature->GetFieldAsString(iField) );
}
```



In C :

```
OGRFeatureDefnH hFDefn = OGR_L_GetLayerDefn(hLayer);
int iField;

for( iField = 0; iField < OGR_FD_GetFieldCount(hFDefn); iField++ )
{
    OGRFieldDefnH hFieldDefn = OGR_FD_GetFieldDefn( hFDefn, iField );

    if( OGR_Fld_GetType(hFieldDefn) == OFTInteger )
        printf( "%d,", OGR_F_GetFieldAsInteger( hFeature, iField ) );
    else if( OGR_Fld_GetType(hFieldDefn) == OFTReal )
        printf( "%.3f,", OGR_F_GetFieldAsDouble( hFeature, iField ) );
    else if( OGR_Fld_GetType(hFieldDefn) == OFTString )
        printf( "%s,", OGR_F_GetFieldAsString( hFeature, iField ) );
    else
        printf( "%s,", OGR_F_GetFieldAsString( hFeature, iField ) );
}
```

There are a few more field types than those explicitly handled above, but a reasonable representation of them can be fetched with the **OGRFeature::GetFieldAsString()** (p. ??) method. In fact we could shorten the above by using **OGRFeature::GetFieldAsString()** (p. ??) for all the types.

Next we want to extract the geometry from the feature, and write out the point geometry x and y. Geometries are returned as a generic **OGRGeometry** (p. ??) pointer. We then determine the specific geometry type, and if it is a point, we cast it to point and operate on it. If it is something else we write placeholders.

In C++ :

```
OGRGeometry *poGeometry;

poGeometry = poFeature->GetGeometryRef();
if( poGeometry != NULL
    && wkbFlatten(poGeometry->getGeometryType()) == wkbPoint )
{
    OGRPoint *poPoint = (OGRPoint *) poGeometry;

    printf( "%.3f,%.3f\n", poPoint->getX(), poPoint->getY() );
}
else
{
    printf( "no point geometry\n" );
}
```

In C :

```
OGRGeometryH hGeometry;

hGeometry = OGR_F_GetGeometryRef(hFeature);
if( hGeometry != NULL
    && wkbFlatten(OGR_G_GetGeometryType(hGeometry)) == wkbPoint )
{
    printf( "%.3f,%.3f\n", OGR_G_GetX(hGeometry, 0), OGR_G_GetY(hGeometry, 0) );
}
else
{
    printf( "no point geometry\n" );
}
```

The `wkbFlatten()` macro is used above to convert the type for a `wkbPoint25D` (a point with a z coordinate) into the base 2D geometry type code (`wkbPoint`). For each 2D geometry type there is a corresponding 2.5D type code. The 2D and 2.5D geometry cases are handled by the same C++ class, so our code will handle 2D or 3D cases properly.

Note that **OGRFeature::GetGeometryRef()** (p. ??) returns a pointer to the internal geometry owned by the **OGRFeature** (p. ??). There we don't actually deleted the return geometry. However, the **OGRLayer::GetNextFeature()** (p. ??) method returns a copy of the feature that is now owned by us. So at the end of use we must free the feature. We could just "delete" it, but this can cause problems in windows builds where the GDAL DLL has a different "heap" from the main program. To be on the safe side we use a GDAL function to delete the feature.

In C++ :

```
OGRFeature::DestroyFeature( poFeature );
}
```

In C :

```
OGR_F_Destroy( hFeature );
}
```

The **OGRLayer** (p. ??) returned by **OGRDataSource::GetLayerByName()** (p. ??) is also a reference to an internal layer owned by the **OGRDataSource** (p. ??) so we don't need to delete it. But we do need to delete the datasource in order to close the input file. Once again we do this with a custom delete method to avoid special win32 heap issues.

In C++ :

```
OGRDataSource::DestroyDataSource( poDS );
}
```

In C :

```
OGR_DS_Destroy( hDS );
}
```

All together our program looks like this.

In C++ :

```
#include "ogr_sfrmts.h"

int main()
{
    OGRRegisterAll();

    OGRDataSource      *poDS;

    poDS = OGRSFDriverRegistrar::Open( "point.shp", FALSE );
    if( poDS == NULL )
    {
        printf( "Open failed.\n" );
        exit( 1 );
    }

    OGRLayer *poLayer;

    poLayer = poDS->GetLayerByName( "point" );

    OGRFeature *poFeature;

    poLayer->ResetReading();
    while( (poFeature = poLayer->GetNextFeature()) != NULL )
    {
        OGRFeatureDefn *poFDefn = poLayer->GetLayerDefn();
        int iField;

        for( iField = 0; iField < poFDefn->GetFieldCount(); iField++ )
        {
            OGRFieldDefn *poFieldDefn = poFDefn->GetFieldDefn( iField );

            if( poFieldDefn->GetType() == OFTInteger )
                printf( "%d,", poFeature->GetFieldAsInteger( iField ) );
            else if( poFieldDefn->GetType() == OFTReal )
                printf( "%.3f,", poFeature->GetFieldAsDouble(iField) );
            else if( poFieldDefn->GetType() == OFTString )
                printf( "%s,", poFeature->GetFieldAsString(iField) );
            else
                printf( "%s,", poFeature->GetFieldAsString(iField) );
        }

        OGRGeometry *poGeometry;

        poGeometry = poFeature->GetGeometryRef();
        if( poGeometry != NULL
            && wkbFlatten(poGeometry->getGeometryType()) == wkbPoint )
        {
            OGRPoint *poPoint = (OGRPoint *) poGeometry;

            printf( "%.3f,%.3f\n", poPoint->getX(), poPoint->getY() );
        }
        else
        {

```

```

        printf( "no point geometry\n" );
    }
    OGRFeature::DestroyFeature( poFeature );
}

OGRDataSource::DestroyDataSource( poDS );
}

```

In C :

```

#include "ogr_api.h"

int main()
{
    OGRRegisterAll();

    OGRDataSourceH hDS;
    OGRLayerH hLayer;
    OGRFeatureH hFeature;

    hDS = OGROpen( "point.shp", FALSE, NULL );
    if( hDS == NULL )
    {
        printf( "Open failed.\n" );
        exit( 1 );
    }

    hLayer = OGR_DS_GetLayerByName( hDS, "point" );

    OGR_L_ResetReading(hLayer);
    while( (hFeature = OGR_L_GetNextFeature(hLayer)) != NULL )
    {
        OGRFeatureDefnH hFDefn;
        int iField;
        OGRGeometryH hGeometry;

        hFDefn = OGR_L_GetLayerDefn(hLayer);

        for( iField = 0; iField < OGR_FD_GetFieldCount(hFDefn); iField++ )
        {
            OGRFieldDefnH hFieldDefn = OGR_FD_GetFieldDefn( hFDefn, iField );

            if( OGR_Fld_GetType(hFieldDefn) == OFTInteger )
                printf( "%d,", OGR_F_GetFieldAsInteger( hFeature, iField ) );
            else if( OGR_Fld_GetType(hFieldDefn) == OFTReal )
                printf( "%.3f,", OGR_F_GetFieldAsDouble( hFeature, iField ) );
            else if( OGR_Fld_GetType(hFieldDefn) == OFTString )
                printf( "%s,", OGR_F_GetFieldAsString( hFeature, iField ) );
            else
                printf( "%s,", OGR_F_GetFieldAsString( hFeature, iField ) );
        }

        hGeometry = OGR_F_GetGeometryRef(hFeature);
        if( hGeometry != NULL
            && wkbFlatten(OGR_G_GetGeometryType(hGeometry)) == wkbPoint )
        {
            printf( "%.3f,%.3f\n", OGR_G_GetX(hGeometry, 0), OGR_G_GetY(hGeometry, 0) );
        }
        else
        {
            printf( "no point geometry\n" );
        }

        OGR_F_Destroy( hFeature );
    }

    OGR_DS_Destroy( hDS );
}

```

In Python:

```

import sys
import ogr

ds = ogr.Open( "point.shp" )
if ds is None:
    print "Open failed.\n"
    sys.exit( 1 )

lyr = ds.GetLayerByName( "point" )

lyr.ResetReading()

```

```

for feat in lyr:

    feat_defn = lyr.GetLayerDefn()
    for i in range(feat_defn.GetFieldCount()):
        field_defn = feat_defn.GetFieldDefn(i)

        # Tests below can be simplified with just :
        # print feat.GetField(i)
        if field_defn.GetType() == ogr.OFTInteger:
            print "%d" % feat.GetFieldAsInteger(i)
        elif field_defn.GetType() == ogr.OFTReal:
            print "%.3f" % feat.GetFieldAsDouble(i)
        elif field_defn.GetType() == ogr.OFTString:
            print "%s" % feat.GetFieldAsString(i)
        else:
            print "%s" % feat.GetFieldAsString(i)

    geom = feat.GetGeometryRef()
    if geom is not None and geom.GetGeometryType() == ogr.wkbPoint:
        print "%.3f, %.3f" % ( geom.GetX(), geom.GetY() )
    else:
        print "no point geometry\n"

ds = None

```

## 2.2 Writing To OGR

As an example of writing through OGR, we will do roughly the opposite of the above. A short program that reads comma separated values from input text will be written to a point shapefile via OGR.

As usual, we start by registering all the drivers, and then fetch the Shapefile driver as we will need it to create our output file.

In C++ :

```

#include "ogr_sfrmts.h"

int main()
{
    const char *pszDriverName = "ESRI Shapefile";
    OGRSFDriver *poDriver;

    OGRRegisterAll();

    poDriver = OGRSFDriverRegistrar::GetRegistrar()->GetDriverByName(
        pszDriverName );
    if( poDriver == NULL )
    {
        printf( "%s driver not available.\n", pszDriverName );
        exit( 1 );
    }
}

```

In C :

```

#include "ogr_api.h"

int main()
{
    const char *pszDriverName = "ESRI Shapefile";
    OGRSFDriverH hDriver;

    OGRRegisterAll();

    hDriver = OGRGetDriverByName( pszDriverName );
    if( hDriver == NULL )
    {
        printf( "%s driver not available.\n", pszDriverName );
        exit( 1 );
    }
}

```

Next we create the datasource. The ESRI Shapefile driver allows us to create a directory full of shapefiles, or a single shapefile as a datasource. In this case we will explicitly create a single file by including the extension in the name. Other drivers behave differently. The second argument to the call is a list of option values, but we will just be using defaults in this case. Details of the options supported are also format specific.

In C ++ :

```
OGRDataSource *poDS;

poDS = poDriver->CreateDataSource( "point_out.shp", NULL );
if( poDS == NULL )
{
    printf( "Creation of output file failed.\n" );
    exit( 1 );
}
```

In C :

```
OGRDataSourceH hDS;

hDS = OGR_Dr_CreateDataSource( hDriver, "point_out.shp", NULL );
if( hDS == NULL )
{
    printf( "Creation of output file failed.\n" );
    exit( 1 );
}
```

Now we create the output layer. In this case since the datasource is a single file, we can only have one layer. We pass `wkbPoint` to specify the type of geometry supported by this layer. In this case we aren't passing any coordinate system information or other special layer creation options.

In C++ :

```
OGRLayer *poLayer;

poLayer = poDS->CreateLayer( "point_out", NULL, wkbPoint, NULL );
if( poLayer == NULL )
{
    printf( "Layer creation failed.\n" );
    exit( 1 );
}
```

In C :

```
OGRLayerH hLayer;

hLayer = OGR_DS_CreateLayer( hDS, "point_out", NULL, wkbPoint, NULL );
if( hLayer == NULL )
{
    printf( "Layer creation failed.\n" );
    exit( 1 );
}
```

Now that the layer exists, we need to create any attribute fields that should appear on the layer. Fields must be added to the layer before any features are written. To create a field we initialize an **OGRField** (p. ??) object with the information about the field. In the case of Shapefiles, the field width and precision is significant in the creation of the output .dbf file, so we set it specifically, though generally the defaults are OK. For this example we will just have one attribute, a name string associated with the x,y point.

Note that the template **OGRField** (p. ??) we pass to `CreateField()` is copied internally. We retain ownership of the object.

In C++:

```
OGRFieldDefn oField( "Name", OFTString );

oField.SetWidth(32);

if( poLayer->CreateField( &oField ) != OGRERR_NONE )
{
    printf( "Creating Name field failed.\n" );
    exit( 1 );
}
```

In C:

```
OGRFieldDefnH hFieldDefn;

hFieldDefn = OGR_Fld_Create( "Name", OFTString );
```

```
OGR_Fld_SetWidth( hFieldDefn, 32);

if( OGR_L_CreateField( hLayer, hFieldDefn, TRUE ) != OGRERR_NONE )
{
    printf( "Creating Name field failed.\n" );
    exit( 1 );
}

OGR_Fld_Destroy( hFieldDefn);
```

The following snippets loops reading lines of the form "x,y,name" from stdin, and parsing them.

In C++ and in C :

```
double x, y;
char szName[33];

while( !feof(stdin)
        && fscanf( stdin, "%lf,%lf,%32s", &x, &y, szName ) == 3 )
{
```

To write a feature to disk, we must create a local **OGRFeature** (p. ??), set attributes and attach geometry before trying to write it to the layer. It is imperative that this feature be instantiated from the **OGRFeatureDefn** (p. ??) associated with the layer it will be written to.

In C++ :

```
OGRFeature *poFeature;

poFeature = OGRFeature::CreateFeature( poLayer->GetLayerDefn() );
poFeature->SetField( "Name", szName );
```

In C :

```
OGRFeatureH hFeature;

hFeature = OGR_F_Create( OGR_L_GetLayerDefn( hLayer ) );
OGR_F_SetFieldString( hFeature, OGR_F_GetFieldIndex(hFeature, "Name"), szName );
```

We create a local geometry object, and assign its copy (indirectly) to the feature. The **OGRFeature::SetGeometry-Directly()** (p. ??) differs from **OGRFeature::SetGeometry()** (p. ??) in that the direct method gives ownership of the geometry to the feature. This is generally more efficient as it avoids an extra deep object copy of the geometry.

In C++ :

```
OGRPoint pt;
pt.setX( x );
pt.setY( y );

poFeature->SetGeometry( &pt );
```

In C :

```
OGRGeometryH hPt;
hPt = OGR_G_CreateGeometry(wkbPoint);
OGR_G_SetPoint_2D(hPt, 0, x, y);

OGR_F_SetGeometry( hFeature, hPt );
OGR_G_DestroyGeometry(hPt);
```

Now we create a feature in the file. The **OGRLayer::CreateFeature()** (p. ??) does not take ownership of our feature so we clean it up when done with it.

In C++ :

```
if( poLayer->CreateFeature( poFeature ) != OGRERR_NONE )
{
    printf( "Failed to create feature in shapefile.\n" );
    exit( 1 );
}

OGRFeature::DestroyFeature( poFeature );
}
```

In C :

```

    if( OGR_L_CreateFeature( hLayer, hFeature ) != OGRERR_NONE )
    {
        printf( "Failed to create feature in shapefile.\n" );
        exit( 1 );
    }

    OGR_F_Destroy( hFeature );
}

```

Finally we need to close down the datasource in order to ensure headers are written out in an orderly way and all resources are recovered.

In C++ :

```

OGRDataSource::DestroyDataSource( poDS );
}

```

In C :

```

OGR_DS_Destroy( hDS );
}

```

The same program all in one block looks like this:

In C++ :

```

#include "ogrsg_frmts.h"

int main()
{
    const char *pszDriverName = "ESRI Shapefile";
    OGRSFDriver *poDriver;

    OGRRegisterAll();

    poDriver = OGRSFDriverRegistrar::GetRegistrar()->GetDriverByName(
        pszDriverName );
    if( poDriver == NULL )
    {
        printf( "%s driver not available.\n", pszDriverName );
        exit( 1 );
    }

    OGRDataSource *poDS;

    poDS = poDriver->CreateDataSource( "point_out.shp", NULL );
    if( poDS == NULL )
    {
        printf( "Creation of output file failed.\n" );
        exit( 1 );
    }

    OGRLayer *poLayer;

    poLayer = poDS->CreateLayer( "point_out", NULL, wkbPoint, NULL );
    if( poLayer == NULL )
    {
        printf( "Layer creation failed.\n" );
        exit( 1 );
    }

    OGRFieldDefn oField( "Name", OFTString );

    oField.SetWidth(32);

    if( poLayer->CreateField( oField ) != OGRERR_NONE )
    {
        printf( "Creating Name field failed.\n" );
        exit( 1 );
    }

    double x, y;
    char szName[33];

    while( !feof(stdin)
        && fscanf( stdin, "%lf,%lf,%32s", &x, &y, szName ) == 3 )
    {

```

```

    OGRFeature *poFeature;

    poFeature = OGRFeature::CreateFeature( poLayer->GetLayerDefn() );
    poFeature->SetField( "Name", szName );

    OGRPoint pt;

    pt.setX( x );
    pt.setY( y );

    poFeature->SetGeometry( &pt );

    if( poLayer->CreateFeature( poFeature ) != OGRERR_NONE )
    {
        printf( "Failed to create feature in shapefile.\n" );
        exit( 1 );
    }

    OGRFeature::DestroyFeature( poFeature );
}

OGRDataSource::DestroyDataSource( poDS );
}

```

In C:

```

#include "ogr_api.h"

int main()
{
    const char *pszDriverName = "ESRI Shapefile";
    OGRSFDriverH hDriver;
    OGRDataSourceH hDS;
    OGRLayerH hLayer;
    OGRFieldDefnH hFieldDefn;
    double x, y;
    char szName[33];

    OGRRegisterAll();

    hDriver = OGRGetDriverByName( pszDriverName );
    if( hDriver == NULL )
    {
        printf( "%s driver not available.\n", pszDriverName );
        exit( 1 );
    }

    hDS = OGR_Dr_CreateDataSource( hDriver, "point_out.shp", NULL );
    if( hDS == NULL )
    {
        printf( "Creation of output file failed.\n" );
        exit( 1 );
    }

    hLayer = OGR_DS_CreateLayer( hDS, "point_out", NULL, wkbPoint, NULL );
    if( hLayer == NULL )
    {
        printf( "Layer creation failed.\n" );
        exit( 1 );
    }

    hFieldDefn = OGR_Fld_Create( "Name", OFTString );
    OGR_Fld_SetWidth( hFieldDefn, 32 );

    if( OGR_L_CreateField( hLayer, hFieldDefn, TRUE ) != OGRERR_NONE )
    {
        printf( "Creating Name field failed.\n" );
        exit( 1 );
    }

    OGR_Fld_Destroy( hFieldDefn );

    while( !feof(stdin)
        && fscanf( stdin, "%lf,%lf,%32s", &x, &y, szName ) == 3 )
    {
        OGRFeatureH hFeature;
        OGRGeometryH hPt;

        hFeature = OGR_F_Create( OGR_L_GetLayerDefn( hLayer ) );
        OGR_F_SetFieldString( hFeature, OGR_F_GetFieldIndex(hFeature, "Name"), szName );

        hPt = OGR_G_CreateGeometry(wkbPoint);
        OGR_G_SetPoint_2D(hPt, 0, x, y);
    }
}

```



```

    OGR_F_SetGeometry( hFeature, hPt );
    OGR_G_DestroyGeometry(hPt);

    if( OGR_L_CreateFeature( hLayer, hFeature ) != OGRERR_NONE )
    {
        printf( "Failed to create feature in shapefile.\n" );
        exit( 1 );
    }

    OGR_F_Destroy( hFeature );
}

OGR_DS_Destroy( hDS );
}

```

### In Python :

```

import sys
import ogr
import string

driverName = "ESRI Shapefile"
drv = ogr.GetDriverByName( driverName )
if drv is None:
    print "%s driver not available.\n" % driverName
    sys.exit( 1 )

ds = drv.CreateDataSource( "point_out.shp" )
if ds is None:
    print "Creation of output file failed.\n"
    sys.exit( 1 )

lyr = ds.CreateLayer( "point_out", None, ogr.wkbPoint )
if lyr is None:
    print "Layer creation failed.\n"
    sys.exit( 1 )

field_defn = ogr.FieldDefn( "Name", ogr.OFTString )
field_defn.SetWidth( 32 )

if lyr.CreateField( field_defn ) != 0:
    print "Creating Name field failed.\n"
    sys.exit( 1 )

# Expected format of user input: x y name
linestring = raw_input()
linelist = string.split(linestring)

while len(linelist) == 3:
    x = float(linelist[0])
    y = float(linelist[1])
    name = linelist[2]

    feat = ogr.Feature( lyr.GetLayerDefn() )
    feat.SetField( "Name", name )

    pt = ogr.Geometry(ogr.wkbPoint)
    pt.SetPoint_2D(0, x, y)

    feat.SetGeometry(pt)

    if lyr.CreateFeature(feat) != 0:
        print "Failed to create feature in shapefile.\n"
        sys.exit( 1 )

    feat.Destroy()

    linestring = raw_input()
    linelist = string.split(linestring)

ds = None

```



## Chapter 3

# OGR Architecture

This document is intended to document the OGR classes. The OGR classes are intended to be generic (not specific to OLE DB or COM or Windows) but are used as a foundation for implementing OLE DB Provider support, as well as client side support for SFCOM. It is intended that these same OGR classes could be used by an implementation of SFCORBA for instance or used directly by C++ programs wanting to use an OpenGIS simple features inspired API.

Because OGR is modelled on the OpenGIS simple features data model, it is very helpful to review the SF-COM, or other simple features interface specifications which can be retrieved from the [Open Geospatial Consortium](#) web site. Data types, and method names are modelled on those from the interface specifications.

### 3.1 Class Overview

- **Geometry** (`ogr_geometry.h`): The geometry classes (**OGRGeometry** (p. ??), etc) encapsulate the OpenGIS model vector data as well as providing some geometry operations, and translation to/from well known binary and text format. A geometry includes a spatial reference system (projection).
- **Spatial Reference** (`ogr_spatialref.h`): An **OGRSpatialReference** (p. ??) encapsulates the definition of a projection and datum.
- **Feature** (`ogr_feature.h`): The **OGRFeature** (p. ??) encapsulates the definition of a whole feature, that is a geometry and a set of attributes.
- **Feature Class Definition** (`ogr_feature.h`): The **OGRFeatureDefn** (p. ??) class captures the schema (set of field definitions) for a group of related features (normally a whole layer).
- **Layer** (`ogr_sf_frmts.h`): **OGRLayer** (p. ??) is an abstract base class represent a layer of features in an **OGRDataSource** (p. ??).
- **Data Source** (`ogr_sf_frmts.h`): An **OGRDataSource** (p. ??) is an abstract base class representing a file or database containing one or more **OGRLayer** (p. ??) objects.
- **Drivers** (`ogr_sf_frmts.h`): An **OGRSFDriver** (p. ??) represents a translator for a specific format, opening **OGRDataSource** (p. ??) objects. All available drivers are managed by the **OGRSFDriverRegistrar** (p. ??).

### 3.2 Geometry

The geometry classes are represent various kinds of vector geometry. All the geometry classes derived from **OGRGeometry** (p. ??) which defines the common services of all geometries. Types of geometry include **OGRPoint** (p. ??), **OGRLineString** (p. ??), **OGRPolygon** (p. ??), **OGRGeometryCollection** (p. ??), **OGRMultiPolygon** (p. ??), **OGRMultiPoint** (p. ??), and **OGRMultiLineString** (p. ??).

Additional intermediate abstract base classes contain functionality that could eventually be implemented by other geometry types. These include **OGRCurve** (p. ??) (base class for **OGRLineString** (p. ??)) and **OGRSurface** (p. ??) (base class for **OGRPolygon** (p. ??)). Some intermediate interfaces modelled in the simple features abstract model and SFCOM are not modelled in OGR at this time. In most cases the methods are aggregated into other classes. This may change.

The **OGRGeometryFactory** (p. ??) is used to convert well known text, and well known binary format data into geometries. These are predefined ASCII and binary formats for representing all the types of simple features geometries.

In a manner based on the geometry object in SFCOM, the **OGRGeometry** (p. ??) includes a reference to an **OGRSpatialReference** (p. ??) object, defining the spatial reference system of that geometry. This is normally a reference to a shared spatial reference object with reference counting for each of the **OGRGeometry** (p. ??) objects using it.

Many of the spatial analysis methods (such as computing overlaps and so forth) are not implemented at this time for **OGRGeometry** (p. ??).

While it is theoretically possible to derive other or more specific geometry classes from the existing **OGRGeometry** (p. ??) classes, this isn't an aspect that has been well thought out. In particular, it would be possible to create specialized classes using the **OGRGeometryFactory** (p. ??) without modifying it.

### 3.3 Spatial Reference

The **OGRSpatialReference** (p. ??) class is intended to store an OpenGIS Spatial Reference System definition. Currently local, geographic and projected coordinate systems are supported. Vertical coordinate systems, geocentric coordinate systems, and compound (horizontal + vertical) coordinate systems are not supported.

The spatial coordinate system data model is inherited from the OpenGIS **Well Known Text** format. A simple form of this is defined in the Simple Features specifications. A more sophisticated form is found in the Coordinate Transformation specification. The **OGRSpatialReference** (p. ??) is built on the features of the Coordinate Transformation specification but is intended to be compatible with the earlier simple features form.

There is also an associated **OGRCoordinateTransformation** (p. ??) class that encapsulates use of PROJ.4 for converting between different coordinate systems. There is a [tutorial](#) available describing how to use the **OGRSpatialReference** (p. ??) class.

### 3.4 Feature / Feature Definition

The **OGRGeometry** (p. ??) captures the geometry of a vector feature ... the spatial position/region of a feature. The **OGRFeature** (p. ??) contains this geometry, and adds feature attributes, feature id, and a feature class identify.

The set of attributes, their types, names and so forth is represented via the **OGRFeatureDefn** (p. ??) class. One **OGRFeatureDefn** (p. ??) normally exists for a layer of features. The same definition is shared in a reference counted manner by the feature of that type (or feature class).

The feature id (FID) of a feature is intended to be a unique identifier for the feature within the layer it is a member of. Freestanding features, or features not yet written to a layer may have a null (**OGRNullFID**) feature id. The feature ids are modelled in OGR as a long integer; however, this is not sufficiently expressive to model the natural feature ids in some formats. For instance, the GML feature id is a string, and the row id in Oracle is larger than 4 bytes.

The feature class also contains an indicator of the types of geometry allowed for that feature class (returned as an **OGRwkbGeometryType** from **OGRFeatureDefn::GetGeomType()** (p. ??)). If this is **wkbUnknown** then any type of geometry is allowed. This implies that features in a given layer can potentially be of different geometry types though they will always share a common attribute schema.

The **OGRFeatureDefn** (p. ??) also contains a concept of default spatial reference system for all features of that type and a feature class name (normally used as a layer name).

## 3.5 Layer

An **OGRLayer** (p. ??) represents a layer of features within a data source. All features in an **OGRLayer** (p. ??) share a common schema and are of the same **OGRFeatureDefn** (p. ??). An **OGRLayer** (p. ??) class also contains methods for reading features from the data source. The **OGRLayer** (p. ??) can be thought of as a gateway for reading and writing features from an underlying data source, normally a file format. In SFCOM and other table based simple features implementation an **OGRLayer** (p. ??) represents a spatial table.

The **OGRLayer** (p. ??) includes methods for sequential and random reading and writing. Read access (via the **OGRLayer::GetNextFeature()** (p. ??) method) normally reads all features, one at a time sequentially; however, it can be limited to return features intersecting a particular geographic region by installing a spatial filter on the **OGRLayer** (p. ??) (via the **OGRLayer::SetSpatialFilter()** (p. ??) method).

One flaw in the current OGR architecture is that the spatial filter is set directly on the **OGRLayer** (p. ??) which is intended to be the only representative of a given layer in a data source. This means it isn't possible to have multiple read operations active at one time with different spatial filters on each. This aspect may be revised in the future to introduce an **OGRLayerView** class or something similar.

Another question that might arise is why the **OGRLayer** (p. ??) and **OGRFeatureDefn** (p. ??) classes are distinct. An **OGRLayer** (p. ??) always has a one-to-one relationship to an **OGRFeatureDefn** (p. ??), so why not amalgamate the classes. There are two reasons:

1. As defined now **OGRFeature** (p. ??) and **OGRFeatureDefn** (p. ??) don't depend on **OGRLayer** (p. ??), so they can exist independently in memory without regard to a particular layer in a data store.
2. The SF CORBA model does not have a concept of a layer with a single fixed schema the way that the SFCOM and SFSQL models do. The fact that features belong to a feature collection that is potentially not directly related to their current feature grouping may be important to implementing SFCORBA support using OGR.

The **OGRLayer** (p. ??) class is an abstract base class. An implementation is expected to be subclassed for each file format driver implemented. **OGRLayers** are normally owned directly by their **OGRDataSource** (p. ??), and aren't instantiated or destroyed directly.

## 3.6 Data Source

An **OGRDataSource** (p. ??) represents a set of **OGRLayer** (p. ??) objects. This usually represents a single file, set of files, database or gateway. An **OGRDataSource** (p. ??) has a list of **OGRLayers** which it owns but can return references to.

**OGRDataSource** (p. ??) is an abstract base class. An implementation is expected to be subclassed for each file format driver implemented. **OGRDataSource** (p. ??) objects are not normally instantiated directly but rather with the assistance of an **OGRSFDriver** (p. ??). Deleting an **OGRDataSource** (p. ??) closes access to the underlying persistent data source, but does not normally result in deletion of that file.

An **OGRDataSource** (p. ??) has a name (usually a filename) that can be used to reopen the data source with an **OGRSFDriver** (p. ??).

The **OGRDataSource** (p. ??) also has support for executing a datasource specific command, normally a form of SQL. This is accomplished via the **OGRDataSource::ExecuteSQL()** (p. ??) method. While some datasources (such as PostGIS and Oracle) pass the SQL through to an underlying database, OGR also includes support for evaluating a subset of the SQL SELECT statement against any datasource.

## 3.7 Drivers

An **OGRSFDriver** (p. ??) object is instantiated for each file format supported. The **OGRSFDriver** (p. ??) objects are registered with the **OGRSFDriverRegistrar** (p. ??), a singleton class that is normally used to open new data sources.

It is intended that a new **OGRSFDriver** (p. ??) derived class be implemented for each file format to be supported (along with a file format specific **OGRDataSource** (p. ??), and **OGRLayer** (p. ??) classes).

On application startup registration functions are normally called for each desired file format. These functions instantiate the appropriate **OGRSFDriver** (p. ??) objects, and register them with the **OGRSFDriverRegistrar** (p. ??). When a data source is to be opened, the registrar will normally try each **OGRSFDriver** (p. ??) in turn, until one succeeds, returning an **OGRDataSource** (p. ??) object.

It is not intended that the **OGRSFDriverRegistrar** (p. ??) be derived from.

## Chapter 4

# OGR Driver Implementation Tutorial

### 4.1 Overall Approach

In general new formats are added to OGR by implementing format specific drivers with subclasses of **OGRSFDriver** (p. ??), **OGRDataSource** (p. ??) and **OGRLayer** (p. ??). The **OGRSFDriver** (p. ??) subclass is registered with the **OGRSFDriverRegistrar** (p. ??) at runtime.

Before following this tutorial to implement an OGR driver, please review the [OGR Architecture](#) document carefully.

The tutorial will be based on implementing a simple ascii point format.

### 4.2 Contents

1. **Implementing OGRSFDriver** (p. ??)
2. **Basic Read Only Data Source** (p. ??)
3. **Read Only Layer** (p. ??)

### 4.3 Implementing OGRSFDriver

The format specific driver class is implemented as a subclass of **OGRSFDriver** (p. ??). One instance of the driver will normally be created, and registered with the **OGRSFDriverRegistrar()** (p. ??). The instantiation of the driver is normally handled by a global C callable registration function, similar to the following placed in the same file as the driver class.

```
void RegisterOGRSPF()
{
    OGRSFDriverRegistrar::GetRegistrar()->RegisterDriver( new OGRSPFDriver );
}
```

The driver class declaration generally looks something like this for a format with read or read and update access (the `Open()` method), creation support (the `CreateDataSource()` method), and the ability to delete a datasource (the `DeleteDataSource()` method).

```
class OGRSPFDriver : public OGRSFDriver
{
public:
    ~OGRSPFDriver();
```

```

    const char    *GetName();
    OGRDataSource *Open( const char *, int );
    OGRDataSource *CreateDataSource( const char *, char ** );
    OGRErr        DeleteDataSource( const char *pszName );
    int           TestCapability( const char * );
};

```

The constructor generally does nothing. The **OGRSFDriver::GetName()** (p. ??) method returns a static string with the name of the driver. This name is specified on the commandline when creating datasources so it is generally good to keep it short and without any special characters or spaces.

```

OGRSPFDriver::~OGRSPFDriver()
{
}

const char *OGRSPFDriver::GetName()
{
    return "SPF";
}

```

The **Open()** method is called by **OGRSFDriverRegistrar::Open()** (p. ??), or from the C API **OGROpen()** (p. ??). The **OGRSFDriver::Open()** (p. ??) method should quietly return NULL if the passed filename is not of the format supported by the driver. If it is the target format, then a new **OGRDataSource** (p. ??) object for the datasource should be returned.

It is common for the **Open()** method to be delegated to an **Open()** method on the actual format's **OGRDataSource** (p. ??) class.

```

OGRDataSource *OGRSPFDriver::Open( const char * pszFilename, int bUpdate )
{
    OGRSPFDataSource    *poDS = new OGRSPFDataSource();

    if( !poDS->Open( pszFilename, bUpdate ) )
    {
        delete poDS;
        return NULL;
    }
    else
        return poDS;
}

```

In OGR the capabilities of drivers, datasources and layers are determined by calling **TestCapability()** on the various objects with names strings representing specific optional capabilities. For the driver the only two capabilities currently tested for are the ability to create datasources and to delete them. In our first pass as a read only SPF driver, these are both disabled. The default return value for unrecognised capabilities should always be FALSE, and the symbolic #defines for capability names (defined in **ogr\_core.h** (p. ??)) should be used instead of the literal strings to avoid typos.

```

int OGRSPFDriver::TestCapability( const char * pszCap )
{
    if( EQUAL(pszCap, ODrCCreateDataSource) )
        return FALSE;
    else if( EQUAL(pszCap, ODrCDeleteDataSource) )
        return FALSE;
    else
        return FALSE;
}

```

Examples of the **CreateDataSource()** and **DeleteDataSource()** methods are left for the section on creation and update.



## 4.4 Basic Read Only Data Source

We will start implementing a minimal read-only datasource. No attempt is made to optimize operations, and default implementations of many methods inherited from **OGRDataSource** (p. ??) are used.

The primary responsibility of the datasource is to manage the list of layers. In the case of the SPF format a datasource is a single file representing one layer so there is at most one layer. The "name" of a datasource should generally be the name passed to the `Open()` method.

The `Open()` method below is not overriding a base class method, but we have it to implement the open operation delegated by the driver class.

For this simple case we provide a stub `TestCapability()` that returns `FALSE` for all extended capabilities. The `TestCapability()` method is pure virtual, so it does need to be implemented.

```
class OGRSPFDataSource : public OGRDataSource
{
    char                *pszName;

    OGRSPFLayer        **papoLayers;
    int                 nLayers;

public:
    OGRSPFDataSource();
    ~OGRSPFDataSource();

    int                 Open( const char * pszFilename, int bUpdate );

    const char          *GetName() { return pszName; }

    int                 GetLayerCount() { return nLayers; }
    OGRLayer            *GetLayer( int );

    int                 TestCapability( const char * ) { return FALSE; }
};
```

The constructor is a simple initializer to a default state. The `Open()` will take care of actually attaching it to a file. The destructor is responsible for orderly cleanup of layers.

```
OGRSPFDataSource::OGRSPFDataSource()
{
    papoLayers = NULL;
    nLayers = 0;

    pszName = NULL;
}

OGRSPFDataSource::~~OGRSPFDataSource()
{
    for( int i = 0; i < nLayers; i++ )
        delete papoLayers[i];
    CPLFree( papoLayers );

    CPLFree( pszName );
}
```

The `Open()` method is the most important one on the datasource, though in this particular instance it passes most of it's work off to the `OGRSPFLayer` constructor if it believes the file is of the desired format.

Note that `Open()` methods should try and determine that a file isn't of the identified format as efficiently as possible, since many drivers may be invoked with files of the wrong format before the correct driver is reached. In this particular `Open()` we just test the file extension but this is generally a poor way of identifying a file format. If available, checking "magic header values" or something similar is preferable.

In the case of the SPF format, update in place is not supported, so we always fail if `bUpdate` is `FALSE`.

```

int  OGRSPFDataSource::Open( const char *pszFilename, int bUpdate )
{
// -----
//      Does this appear to be an .spf file?
// -----
    if( !EQUAL( CPLGetExtension(pszFilename), "spf" ) )
        return FALSE;

    if( bUpdate )
    {
        CPLError( CE_Failure, CPLE_OpenFailed,
            "Update access not supported by the SPF driver." );
        return FALSE;
    }

// -----
//      Create a corresponding layer.
// -----
    nLayers = 1;
    papoLayers = (OGRSPFLayer **) CPLMalloc(sizeof(void*));

    papoLayers[0] = new OGRSPFLayer( pszFilename );

    pszName = CPLStrdup( pszFilename );

    return TRUE;
}

```

A `GetLayer()` method also needs to be implemented. Since the layer list is created in the `Open()` this is just a lookup with some safety testing.

```

OGRLayer *OGRSPFDataSource::GetLayer( int iLayer )
{
    if( iLayer < 0 || iLayer >= nLayers )
        return NULL;
    else
        return papoLayers[iLayer];
}

```

## 4.5 Read Only Layer

The `OGRSPFLayer` implements layer semantics for an `.spf` file. It provides access to a set of feature objects in a consistent coordinate system with a particular set of attribute columns. Our class definition looks like this:

```

class OGRSPFLayer : public OGRLayer
{
    OGRFeatureDefn      *poFeatureDefn;

    FILE                *fp;

    int                  nNextFID;

public:
    OGRSPFLayer( const char *pszFilename );
    ~OGRSPFLayer();

    void                ResetReading();
    OGRFeature *        GetNextFeature();

    OGRFeatureDefn *    GetLayerDefn() { return poFeatureDefn; }

    int                  TestCapability( const char * ) { return FALSE; }
};

```

The layer constructor is responsible for initialization. The most important initialization is setting up the **OGRFeatureDefn** (p. ??) for the layer. This defines the list of fields and their types, the geometry type and the coordinate system

for the layer. In the SPF format the set of fields is fixed - a single string field and we have no coordinate system info to set.

Pay particular attention to the reference counting of the **OGRFeatureDefn** (p. ??). As **OGRFeature** (p. ??)'s for this layer will also take a reference to this definition it is important that we also establish a reference on behalf of the layer itself.

```
OGRSPFLayer::OGRSPFLayer( const char *pszFilename )
{
    nNextFID = 0;

    poFeatureDefn = new OGRFeatureDefn( CPLGetBasename( pszFilename ) );
    poFeatureDefn->Reference();
    poFeatureDefn->SetGeomType( wkbPoint );

    OGRFieldDefn oFieldTemplate( "Name", OFTString );

    poFeatureDefn->AddFieldDefn( &oFieldTemplate );

    fp = VSIFOpenL( pszFilename, "r" );
    if( fp == NULL )
        return;
}
```

Note that the destructor uses `Release()` on the **OGRFeatureDefn** (p. ??). This will destroy the feature definition if the reference count drops to zero, but if the application is still holding onto a feature from this layer, then that feature will hold a reference to the feature definition and it will not be destroyed here (which is good!).

```
OGRSPFLayer::~~OGRSPFLayer()
{
    poFeatureDefn->Release();
    if( fp != NULL )
        VSIFCloseL( fp );
}
```

The `GetNextFeature()` method is usually the work horse of **OGRLayer** (p. ??) implementations. It is responsible for reading the next feature according to the current spatial and attribute filters installed.

The `while()` loop is present to loop until we find a satisfactory feature. The first section of code is for parsing a single line of the SPF text file and establishing the x, y and name for the line.

```
OGRFeature *OGRSPFLayer::GetNextFeature()
{
    // -----
    // Loop till we find a feature matching our requirements.
    // -----
    while( TRUE )
    {
        const char *pszLine;
        const char *pszName;

        pszLine = CPLReadLineL( fp );

        // Are we at end of file (out of features)?
        if( pszLine == NULL )
            return NULL;

        double dfX;
        double dfY;

        dfX = atof(pszLine);

        pszLine = strstr(pszLine, "|");
        if( pszLine == NULL )
            continue; // we should issue an error!
        else
```

```

        pszLine++;

        dfY = atof(pszLine);

        pszLine = strstr(pszLine, "|");
        if( pszLine == NULL )
            continue; // we should issue an error!
        else
            pszName = pszLine+1;

```

The next section turns the x, y and name into a feature. Also note that we assign a linearly incremented feature id. In our case we started at zero for the first feature, though some drivers start at 1.

```

OGRFeature *poFeature = new OGRFeature( poFeatureDefn );

poFeature->SetGeometryDirectly( new OGRPoint( dfX, dfY ) );
poFeature->SetField( 0, pszName );
poFeature->SetFID( nNextFID++ );

```

Next we check if the feature matches our current attribute or spatial filter if we have them. Methods on the **OGR-Layer** (p. ??) base class support maintain filters in the **OGR-Layer** (p. ??) member fields `m_poFilterGeom` (spatial filter) and `m_poAttrQuery` (attribute filter) so we can just use these values here if they are non-NULL. The following test is essentially "stock" and done the same in all formats. Some formats also do some spatial filtering ahead of time using a spatial index.

If the feature meets our criteria we return it. Otherwise we destroy it, and return to the top of the loop to fetch another to try.

```

        if( (m_poFilterGeom == NULL
            || FilterGeometry( poFeature->GetGeometryRef() ) )
            && (m_poAttrQuery == NULL
            || m_poAttrQuery->Evaluate( poFeature ) ) )
            return poFeature;

        delete poFeature;
    }
}

```

While in the middle of reading a feature set from a layer, or at any other time the application can call `ResetReading()` which is intended to restart reading at the beginning of the feature set. We implement this by seeking back to the beginning of the file, and resetting our feature id counter.

```

void OGRSPFLayer::ResetReading()
{
    VSIFSeekL( fp, 0, SEEK_SET );
    nNextFID = 0;
}

```

In this implementation we do not provide a custom implementation for the `GetFeature()` method. This means an attempt to read a particular feature by it's feature id will result in many calls to `GetNextFeature()` till the desired feature is found. However, in a sequential text format like `spf` there is little else we could do anyway.

There! We have completed a simple read-only feature file format driver.

## Chapter 5

# OGR SQL

The **OGRDataSource** (p. ??) supports executing commands against a datasource via the **OGRDataSource::ExecuteSQL()** (p. ??) method. While in theory any sort of command could be handled this way, in practice the mechanism is used to provide a subset of SQL SELECT capability to applications. This page discusses the generic SQL implementation implemented within OGR, and issue with driver specific SQL support.

Since GDAL/OGR 1.10, an alternate "dialect", the SQLite dialect, can be used instead of the OGRSQL dialect. Refer to the [SQLite SQL dialect](#) page for more details.

The **OGRLayer** (p. ??) class also supports applying an attribute query filter to features returned using the **OGRLayer::SetAttributeFilter()** (p. ??) method. The syntax for the attribute filter is the same as the WHERE clause in the OGR SQL SELECT statement. So everything here with regard to the WHERE clause applies in the context of the SetAttributeFilter() method.

NOTE: OGR SQL has been reimplemented for GDAL/OGR 1.8.0. Many features discussed below, notably arithmetic expressions, and expressions in the field list, were not support in GDAL/OGR 1.7.x and earlier. See RFC 28 for details of the new features in GDAL/OGR 1.8.0.

### 5.1 SELECT

The SELECT statement is used to fetch layer features (analogous to table rows in an RDBMS) with the result of the query represented as a temporary layer of features. The layers of the datasource are analogous to tables in an RDBMS and feature attributes are analogous to column values. The simplest form of OGR SQL SELECT statement looks like this:

```
SELECT * FROM polylayer
```

In this case all features are fetched from the layer named "polylayer", and all attributes of those features are returned. This is essentially equivalent to accessing the layer directly. In this example the "\*" is the list of fields to fetch from the layer, with "\*" meaning that all fields should be fetched.

This slightly more sophisticated form still pulls all features from the layer but the schema will only contain the EAS\_ID and PROP\_VALUE attributes. Any other attributes would be discarded.

```
SELECT eas_id, prop_value FROM polylayer
```

A much more ambitious SELECT, restricting the features fetched with a WHERE clause, and sorting the results might look like:

```
SELECT * from polylayer WHERE prop_value > 220000.0 ORDER BY prop_value DESC
```

This select statement will produce a table with just one feature, with one attribute (named something like "count\_eas\_id") containing the number of distinct values of the eas\_id attribute.

```
SELECT COUNT(DISTINCT eas_id) FROM polylayer
```

### 5.1.1 Field List Operators

The field list is a comma separate list of the fields to be carried into the output features from the source layer. They will appear on output features in the order they appear on in the field list, so the field list may be used to re-order the fields.

A special form of the field list uses the DISTINCT keyword. This returns a list of all the distinct values of the named attribute. When the DISTINCT keyword is used, only one attribute may appear in the field list. The DISTINCT keyword may be used against any type of field. Currently the distinctness test against a string value is case insensitive in OGR SQL. The result of a SELECT with a DISTINCT keyword is a layer with one column (named the same as the field operated on), and one feature per distinct value. Geometries are discarded. The distinct values are assembled in memory, so a lot of memory may be used for datasets with a large number of distinct values.

```
SELECT DISTINCT areacode FROM polylayer
```

There are also several summarization operators that may be applied to columns. When a summarization operator is applied to any field, then all fields must have summarization operators applied. The summarization operators are COUNT (a count of instances), AVG (numerical average), SUM (numerical sum), MIN (lexical or numerical minimum), and MAX (lexical or numerical maximum). This example produces a variety of summarization information on parcel property values:

```
SELECT MIN(prop_value), MAX(prop_value), AVG(prop_value), SUM(prop_value),  
       COUNT(prop_value) FROM polylayer WHERE prov_name = "Ontario"
```

It is also possible to apply the COUNT() operator to a DISTINCT SELECT to get a count of distinct values, for instance:

```
SELECT COUNT(DISTINCT areacode) FROM polylayer
```

Note: prior to OGR 1.9.0, null values were counted in COUNT(column\_name) or COUNT(DISTINCT column\_name), which was not conformant with the SQL standard. Since OGR 1.9.0, only non-null values are counted.

As a special case, the COUNT() operator can be given a "\*" argument instead of a field name which is a short form for count all the records.

```
SELECT COUNT(*) FROM polylayer
```

Field names can also be prefixed by a table name though this is only really meaningful when performing joins. It is further demonstrated in the JOIN section.

Field definitions can also be complex expressions using arithmetic, and functional operators. However, the DISTINCT keyword, and summarization operators like MIN, MAX, AVG and SUM may not be applied to expression fields.

```
SELECT cost+tax from invoice
```

or

```
SELECT CONCAT(owner_first_name,' ',owner_last_name) from properties
```

#### 5.1.1.1 Functions

Starting with OGR 1.8.2, the SUBSTR function can be used to extract a substring from a string. Its syntax is the following one : SUBSTR(string\_expr, start\_offset [, length]). It extracts a substring of string\_expr, starting at offset start\_offset (1 being the first character of string\_expr, 2 the second one, etc...). If start\_offset is a negative value, the substring is extracted from the end of the string (-1 is the last character of the string, -2 the character before the last character, ...). If length is specified, up to length characters are extracted from the string. Otherwise the remainder of the string is extracted.

Note: for the time being, the character is considered to be equivalent to bytes, which may not be appropriate for multi-byte encodings like UTF-8.

```
SELECT SUBSTR('abcdef',1,2) FROM xxx --> 'ab'
SELECT SUBSTR('abcdef',4) FROM xxx --> 'def'
SELECT SUBSTR('abcdef',-2) FROM xxx --> 'ef'
```

#### 5.1.1.2 Using the field name alias

OGR SQL supports renaming the fields following the SQL92 specification by using the AS keyword according to the following example:

```
SELECT *, OGR_STYLE AS STYLE FROM polylayer
```

The field name alias can be used as the last operation in the column specification. Therefore we cannot rename the fields inside an operator, but we can rename whole column expression, like these two:

```
SELECT COUNT(areacode) AS 'count' FROM polylayer
SELECT dollars/100.0 AS cents FROM polylayer
```

#### 5.1.1.3 Changing the type of the fields

Starting with GDAL 1.6.0, OGR SQL supports changing the type of the columns by using the SQL92 compliant CAST operator according to the following example:

```
SELECT *, CAST(OGR_STYLE AS character(255)) FROM rivers
```

Currently casting to the following target types are supported:

1. character(field\_length). By default, field\_length=1.
2. float(field\_length)
3. numeric(field\_length, field\_precision)
4. integer(field\_length)
5. date(field\_length)
6. time(field\_length)
7. timestamp(field\_length)

Specifying the field\_length and/or the field\_precision is optional. An explicit value of zero can be used as the width for character() to indicate variable width. Conversion to the 'integer list', 'double list' and 'string list' OGR data types are not supported, which doesn't conform to the SQL92 specification.

While the CAST operator can be applied anywhere in an expression, including in a WHERE clause, the detailed control of output field format is only supported if the CAST operator is the "outer most" operators on a field in the field definition list. In other contexts it is still useful to convert between numeric, string and date data types.

### 5.1.2 WHERE

The argument to the WHERE clause is a logical expression used select records from the source layer. In addition to its use within the WHERE statement, the WHERE clause handling is also used for OGR attribute queries on regular layers via **OGRLayer::SetAttributeFilter()** (p. ??).

In addition to the arithmetic and other functional operators available in expressions in the field selection clause of the SELECT statement, in the WHERE context logical operators are also available and the evaluated value of the expression should be logical (true or false).

The available logical operators are =, !=, <>, <, >, <=, >=, **LIKE** and **ILIKE**, **BETWEEN** and **IN**. Most of the operators are self explanatory, but it is worth noting that != is the same as <>, the string equality is case insensitive, but the <, >, <= and >= operators are case sensitive. Both the LIKE and ILIKE operators are case insensitive.

The value argument to the **LIKE** operator is a pattern against which the value string is matched. In this pattern percent (%) matches any number of characters, and underscore ( \_ ) matches any one character. An optional ESCAPE escape\_char clause can be added so that the percent or underscore characters can be searched as regular characters, by being preceded with the escape\_char.

String	Pattern	Matches?
-----	-----	-----
Alberta	ALB%	Yes
Alberta	_lberta	Yes
St. Alberta	_lberta	No
St. Alberta	%lberta	Yes
Robarts St.	%Robarts%	Yes
12345	123%45	Yes
123.45	12?45	No
NON 1P0	%NON%	Yes
L4C 5E2	%NON%	No

The **IN** takes a list of values as it's argument and tests the attribute value for membership in the provided set.

Value	Value Set	Matches?
-----	-----	-----
321	IN (456,123)	No
"Ontario"	IN ("Ontario", "BC")	Yes
"Ont"	IN ("Ontario", "BC")	No
1	IN (0,2,4,6)	No

The syntax of the **BETWEEN** operator is "field\_name BETWEEN value1 AND value2" and it is equivalent to "field\_name >= value1 AND field\_name <= value2".

In addition to the above binary operators, there are additional operators for testing if a field is null or not. These are the **IS NULL** and **IS NOT NULL** operators.

Basic field tests can be combined in more complicated predicates using logical operators include **AND**, **OR**, and the unary logical **NOT**. Subexpressions should be bracketed to make precedence clear. Some more complicated predicates are:

```
SELECT * FROM poly WHERE (prop_value >= 100000) AND (prop_value < 200000)
SELECT * FROM poly WHERE NOT (area_code LIKE "NON%")
SELECT * FROM poly WHERE (prop_value IS NOT NULL) AND (prop_value < 100000)
```

### 5.1.3 WHERE Limitations

1. Fields must all come from the primary table (the one listed in the FROM clause).
2. All string comparisons are case insensitive except for <, >, <= and >=.

### 5.1.4 ORDER BY

The **ORDER BY** clause is used force the returned features to be reordered into sorted order (ascending or descending) on one of the field values. Ascending (increasing) order is the default if neither the ASC or DESC keyword is provided. For example:

```
SELECT * FROM property WHERE class_code = 7 ORDER BY prop_value DESC
SELECT * FROM property ORDER BY prop_value
SELECT * FROM property ORDER BY prop_value ASC
SELECT DISTINCT zip_code FROM property ORDER BY zip_code
```

Note that ORDER BY clauses cause two passes through the feature set. One to build an in-memory table of field values corresponded with feature ids, and a second pass to fetch the features by feature id in the sorted order. For formats which cannot efficiently randomly read features by feature id this can be a very expensive operation.

Sorting of string field values is case sensitive, not case insensitive like in most other parts of OGR SQL.



### 5.1.5 JOINS

OGR SQL supports a limited form of one to one JOIN. This allows records from a secondary table to be looked up based on a shared key between it and the primary table being queried. For instance, a table of city locations might include a *nation\_id* column that can be used as a reference into a secondary *nation* table to fetch a nation name. A joined query might look like:

```
SELECT city.*, nation.name FROM city
LEFT JOIN nation ON city.nation_id = nation.id
```

This query would result in a table with all the fields from the city table, and an additional "nation.name" field with the nation name pulled from the nation table by looking for the record in the nation table that has the "id" field with the same value as the city.nation\_id field.

Joins introduce a number of additional issues. One is the concept of table qualifiers on field names. For instance, referring to city.nation\_id instead of just nation\_id to indicate the nation\_id field from the city layer. The table name qualifiers may only be used in the field list, and within the **ON** clause of the join.

Wildcards are also somewhat more involved. All fields from the primary table (*city* in this case) and the secondary table (*nation* in this case) may be selected using the usual \* wildcard. But the fields of just one of the primary or secondary table may be selected by prefixing the asterisk with the table name.

The field names in the resulting query layer will be qualified by the table name, if the table name is given as a qualifier in the field list. In addition field names will be qualified with a table name if they would conflict with earlier fields. For instance, the following select would result might result in a results set with a *name*, *nation\_id*, *nation.nation\_id* and *nation.name* field if the city and nation tables both have the *nation\_id* and *name* fieldnames.

```
SELECT * FROM city LEFT JOIN nation ON city.nation_id = nation.nation_id
```

On the other hand if the nation table had a *continent\_id* field, but the city table did not, then that field would not need to be qualified in the result set. However, if the selected instead looked like the following statement, all result fields would be qualified by the table name.

```
SELECT city.*, nation.* FROM city
LEFT JOIN nation ON city.nation_id = nation.nation_id
```

In the above examples, the *nation* table was found in the same datasource as the *city* table. However, the OGR join support includes the ability to join against a table in a different data source, potentially of a different format. This is indicated by qualifying the secondary table name with a datasource name. In this case the secondary datasource is opened using normal OGR semantics and utilized to access the secondary table until the query result is no longer needed.

```
SELECT * FROM city
LEFT JOIN '/usr2/data/nation.dbf'.nation ON city.nation_id = nation.nation_id
```

While not necessarily very useful, it is also possible to introduce table aliases to simplify some SELECT statements. This can also be useful to disambiguate situations where tables of the same name are being used from different data sources. For instance, if the actual tables names were messy we might want to do something like:

```
SELECT c.name, n.name FROM project_615_city c
LEFT JOIN '/usr2/data/project_615_nation.dbf'.project_615_nation n
ON c.nation_id = n.nation_id
```

It is possible to do multiple joins in a single query.

```
SELECT city.name, prov.name, nation.name FROM city
LEFT JOIN province ON city.prov_id = province.id
LEFT JOIN nation ON city.nation_id = nation.id
```

### 5.1.6 JOIN Limitations

1. Joins can be very expensive operations if the secondary table is not indexed on the key field being used.
2. Joined fields may not be used in WHERE clauses, or ORDER BY clauses at this time. The join is essentially evaluated after all primary table subsetting is complete, and after the ORDER BY pass.
3. Joined fields may not be used as keys in later joins. So you could not use the province id in a city to lookup the province record, and then use a nation id from the province id to lookup the nation record. This is a sensible thing to want and could be implemented, but is not currently supported.
4. Datasource names for joined tables are evaluated relative to the current processes working directory, not the path to the primary datasource.
5. These are not true LEFT or RIGHT joins in the RDBMS sense. Whether or not a secondary record exists for the join key or not, one and only one copy of the primary record is returned in the result set. If a secondary record cannot be found, the secondary derived fields will be NULL. If more than one matching secondary field is found only the first will be used.

## 5.2 UNION ALL

(OGR >= 1.10.0)

The SQL engine can deal with several SELECT combined with UNION ALL. The effect of UNION ALL is to concatenate the rows returned by the right SELECT statement to the rows returned by the left SELECT statement.

```
[()] SELECT field_list FROM first_layer [WHERE where_expr] []]
UNION ALL [()] SELECT field_list FROM second_layer [WHERE where_expr] []]
[UNION ALL [()] SELECT field_list FROM third_layer [WHERE where_expr] []]]*
```

### 5.2.1 UNION ALL restrictions

The processing of UNION ALL in OGR differs from the SQL standard, in which it accepts that the columns from the various SELECT are not identical. In that case, it will return a super-set of all the fields from each SELECT statement.

There is also a restriction : ORDER BY can only be specified for each SELECT, and not at the level of the result of the union.

## 5.3 SPECIAL FIELDS

The OGR SQL query processor treats some of the attributes of the features as built-in special fields can be used in the SQL statements likewise the other fields. These fields can be placed in the select list, the WHERE clause and the ORDER BY clause respectively. The special field will not be included in the result by default but it may be explicitly included by adding it to the select list. When accessing the field values the special fields will take precedence over the other fields with the same names in the data source.

### 5.3.1 FID

Normally the feature id is a special property of a feature and not treated as an attribute of the feature. In some cases it is convenient to be able to utilize the feature id in queries and result sets as a regular field. To do so use the name **FID**. The field wildcard expansions will not include the feature id, but it may be explicitly included using a syntax like:

```
SELECT FID, * FROM nation
```

### 5.3.2 OGR\_GEOMETRY

Some of the data sources (like MapInfo tab) can handle geometries of different types within the same layer. The **OGR\_GEOMETRY** special field represents the geometry type returned by **OGRGeometry::getGeometryName()** (p. ??) and can be used to distinguish the various types. By using this field one can select particular types of the geometries like:

```
SELECT * FROM nation WHERE OGR_GEOMETRY='POINT' OR OGR_GEOMETRY='POLYGON'
```

### 5.3.3 OGR\_GEOM\_WKT

The Well Known Text representation of the geometry can also be used as a special field. To select the WKT of the geometry **OGR\_GEOM\_WKT** might be included in the select list, like:

```
SELECT OGR_GEOM_WKT, * FROM nation
```

Using the **OGR\_GEOM\_WKT** and the **LIKE** operator in the WHERE clause we can get similar effect as using **OGR\_GEOMETRY**:

```
SELECT OGR_GEOM_WKT, * FROM nation WHERE OGR_GEOM_WKT
LIKE 'POINT%' OR OGR_GEOM_WKT LIKE 'POLYGON%'
```

### 5.3.4 OGR\_GEOM\_AREA

(Since GDAL 1.7.0)

The **OGR\_GEOM\_AREA** special field returns the area of the feature's geometry computed by the **OGRSurface::get\_Area()** (p. ??) method. For **OGRGeometryCollection** (p. ??) and **OGRMultiPolygon** (p. ??) the value is the sum of the areas of its members. For non-surface geometries the returned area is 0.0.

For example, to select only polygon features larger than a given area:

```
SELECT * FROM nation WHERE OGR_GEOM_AREA > 10000000'
```

### 5.3.5 OGR\_STYLE

The **OGR\_STYLE** special field represents the style string of the feature returned by **OGRFeature::GetStyleString()** (p. ??). By using this field and the **LIKE** operator the result of the query can be filtered by the style. For example we can select the annotation features as:

```
SELECT * FROM nation WHERE OGR_STYLE LIKE 'LABEL%'
```

## 5.4 CREATE INDEX

Some OGR SQL drivers support creating of attribute indexes. Currently this includes the Shapefile driver. An index accelerates very simple attribute queries of the form *fieldname = value*, which is what is used by the **JOIN** capability. To create an attribute index on the *nation\_id* field of the *nation* table a command like this would be used:

```
CREATE INDEX ON nation USING nation_id
```

### 5.4.1 Index Limitations

1. Indexes are not maintained dynamically when new features are added to or removed from a layer.
2. Very long strings (longer than 256 characters?) cannot currently be indexed.

3. To recreate an index it is necessary to drop all indexes on a layer and then recreate all the indexes.
4. Indexes are not used in any complex queries. Currently the only query the will accelerate is a simple "field = value" query.

## 5.5 DROP INDEX

The OGR SQL DROP INDEX command can be used to drop all indexes on a particular table, or just the index for a particular column.

```
DROP INDEX ON nation USING nation_id
DROP INDEX ON nation
```

## 5.6 ALTER TABLE

(OGR >= 1.9.0)

The following OGR SQL ALTER TABLE commands can be used.

1. "ALTER TABLE tablename ADD [COLUMN] columnname columntype" to add a new field. Supported if the layer declares the OLCCreateField capability.
2. "ALTER TABLE tablename RENAME [COLUMN] oldcolumnname TO newcolumnname" to rename an existing field. Supported if the layer declares the OLCAAlterFieldDefn capability.
3. "ALTER TABLE tablename ALTER [COLUMN] columnname TYPE columntype" to change the type of an existing field. Supported if the layer declares the OLCAAlterFieldDefn capability.
4. "ALTER TABLE tablename DROP [COLUMN] columnname" to delete an existing field. Supported if the layer declares the OLCDeleteField capability.

The columntype value follows the syntax of the types supported by the CAST operator described above.

```
ALTER TABLE nation ADD COLUMN myfield integer
ALTER TABLE nation RENAME COLUMN myfield TO myfield2
ALTER TABLE nation ALTER COLUMN myfield2 TYPE character(15)
ALTER TABLE nation DROP COLUMN myfield2
```

## 5.7 DROP TABLE

(OGR >= 1.9.0)

The OGR SQL DROP TABLE command can be used to delete a table. This is only supported on datasources that declare the ODSCDeleteLayer capability.

```
DROP TABLE nation
```

## 5.8 ExecuteSQL()

SQL is executed against an **OGRDataSource** (p. ??), not against a specific layer. The call looks like this:

```
OGRLayer * OGRDataSource::ExecuteSQL( const char *pszSQLCommand,
                                       OGRGeometry *poSpatialFilter,
                                       const char *pszDialect );
```

The `pszDialect` argument is in theory intended to allow for support of different command languages against a provider, but for now applications should always pass an empty (not NULL) string to get the default dialect.

The `poSpatialFilter` argument is a geometry used to select a bounding rectangle for features to be returned in a manner similar to the **`OGRLayer::SetSpatialFilter()`** (p. ??) method. It may be NULL for no special spatial restriction.

The result of an `ExecuteSQL()` call is usually a temporary **`OGRLayer`** (p. ??) representing the results set from the statement. This is the case for a SELECT statement for instance. The returned temporary layer should be released with `OGRDataSource::ReleaseResultSet()` method when no longer needed. Failure to release it before the datasource is destroyed may result in a crash.

## 5.9 Non-OGR SQL

All OGR drivers for database systems: `MySQL`, `PostgreSQL` and `PostGIS (PG)`, `Oracle (OCI)`, `SQLite`, `ODBC`, `ESRI Personal Geodatabase (PGeo)` and `MS SQL Spatial (MSSQLSpatial)`, override the **`OGRDataSource::ExecuteSQL()`** (p. ??) function with dedicated implementation and, by default, pass the SQL statements directly to the underlying RDBMS. In these cases the SQL syntax varies in some particulars from OGR SQL. Also, anything possible in SQL can then be accomplished for these particular databases. Only the result of SQL WHERE statements will be returned as layers.



## Chapter 6

# SQLite SQL dialect

Since GDAL/OGR 1.10, the SQLite "dialect" can be used as an alternate SQL dialect to the OGR SQL dialect. This assumes that GDAL/OGR is built with support for SQLite ( $\geq 3.6$ ), and preferably with Spatialite support too to benefit from spatial functions.

The SQLite dialect may be used with any OGR datasource, like the OGR SQL dialect. It is available through the **OGRDataSource::ExecuteSQL()** (p. ??) method by specifying the pszDialect to "SQLITE". For the `ogrinfo` or `ogr2ogr` utility, you must specify the "-dialect SQLITE" option.

This is mainly aimed to execute SELECT statements, but, for datasources that support update, INSERT/UPDATE/-DELETE statements can also be run.

The syntax of the SQL statements is fully the one of the SQLite SQL engine. You can refer to the following pages:

- [SELECT documentation](#)
- [INSERT documentation](#)
- [UPDATE documentation](#)
- [DELETE documentation](#)

### 6.1 SELECT statement

The SELECT statement is used to fetch layer features (analogous to table rows in an RDBMS) with the result of the query represented as a temporary layer of features. The layers of the datasource are analogous to tables in an RDBMS and feature attributes are analogous to column values. The simplest form of OGR SQLITE SELECT statement looks like this:

```
SELECT * FROM polylayer
```

More complex statements can of course be used, including WHERE, JOIN, USING, GROUP BY, ORDER BY, sub SELECT, ...

The table names that can be used are the layer names available in the datasource on which the ExecuteSQL() method is called.

Similarly to OGRSQL, it is also possible to refer to layers of other datasources with the following syntax : "other\_  
datasource\_name"."layer\_name".

```
SELECT p.*, NAME FROM poly p JOIN "idlink.dbf"."idlink" il USING (eas_id)
```

The column names that can be used in the result column list, in WHERE, JOIN, ... clauses are the field names of the layers. Expressions, SQLite functions can also be used, spatial functions, etc...

The conditions on fields expressed in WHERE clauses, or in JOINS are translated, as far as possible, as attribute filters that are applied on the underlying OGR layers. Joins can be very expensive operations if the secondary table is not indexed on the key field being used.

### 6.1.1 Geometry field

The **GEOMETRY** special field represents the geometry of the feature returned by **OGRFeature::GetGeometryRef()** (p. ??). It can be explicitly specified in the result column list of a SELECT, and is automatically selected if the wildcard is used.

For OGR layers that have a non-empty geometry column name (generally for RDBMS datasources), as returned by **OGRLayer::GetGeometryColumn()** (p. ??), the name of the geometry special field in the SQL statement will be the name of the geometry column of the underlying OGR layer.

```
SELECT EAS_ID, GEOMETRY FROM poly

returns:

OGRFeature(SELECT):0
  EAS_ID (Real) = 168
  POLYGON ((479819.84375 4765180.5,479690.1875 4765259.5,[...],479819.84375 4765180.5))

SELECT * FROM poly

returns:

OGRFeature(SELECT):0
  AREA (Real) = 215229.266
  EAS_ID (Real) = 168
  PRFEDEA (String) = 35043411
  POLYGON ((479819.84375 4765180.5,479690.1875 4765259.5,[...],479819.84375 4765180.5))
```

### 6.1.2 OGR\_STYLE special field

The **OGR\_STYLE** special field represents the style string of the feature returned by **OGRFeature::GetStyleString()** (p. ??). By using this field and the **LIKE** operator the result of the query can be filtered by the style. For example we can select the annotation features as:

```
SELECT * FROM nation WHERE OGR_STYLE LIKE 'LABEL%'
```

### 6.1.3 Spatialite SQL functions

When GDAL/OGR is build with support for the Spatialite library, a lot of extra SQL functions, in particular spatial functions, can be used in results column fields, WHERE clauses, etc....

```
SELECT EAS_ID, ST_Area(GEOMETRY) AS area FROM poly WHERE
  ST_Intersects(GEOMETRY, BuildCircleMbr(479750.6875,4764702.0,100))

returns:

OGRFeature(SELECT):0
  EAS_ID (Real) = 169
  area (Real) = 101429.9765625

OGRFeature(SELECT):1
  EAS_ID (Real) = 165
  area (Real) = 596610.3359375

OGRFeature(SELECT):2
  EAS_ID (Real) = 170
  area (Real) = 5268.8125
```

### 6.1.4 OGR datasource SQL functions

The **ogr\_datasource\_load\_layers(datasource\_name[, update\_mode[, prefix]])** function can be used to automatically load all the layers of a datasource as **VirtualOGR** tables.



```
sqlite> SELECT load_extension('libgdal.so');

sqlite> SELECT load_extension('libspatialite.so');

sqlite> SELECT ogr_datasource_load_layers('poly.shp');
1
sqlite> SELECT * FROM sqlite_master;
table|poly|poly|0|CREATE VIRTUAL TABLE "poly" USING VirtualOGR('poly.shp', 0, 'poly')
```

### 6.1.5 OGR layer SQL functions

The following SQL functions are available and operate on a layer name : **ogr\_layer\_Extent()**, **ogr\_layer\_SRID()**, **ogr\_layer\_GeometryType()** and **ogr\_layer\_FeatureCount()**

```
SELECT ogr_layer_Extent('poly'), ogr_layer_SRID('poly') AS srid,
       ogr_layer_GeometryType('poly') AS geomtype, ogr_layer_FeatureCount('poly') AS count

returns:

OGRFeature(SELECT):0
  srid (Integer) = 40004
  geomtype (String) = POLYGON
  count (Integer) = 10
  POLYGON ((478315.53125 4762880.5,481645.3125 4762880.5,481645.3125 4765610.5,478315.53125 4765610.5,47831
5.53125 4762880.5))
```

### 6.1.6 OGR compression functions

**ogr\_deflate(text\_or\_blob[, compression\_level])** returns a binary blob compressed with the ZLib deflate algorithm. See **CPLZLibDeflate()** (p. ??)

**ogr\_inflate(compressed\_blob)** returns the decompressed binary blob, from a blob compressed with the ZLib deflate algorithm. If the decompressed binary is a string, use **CAST(ogr\_inflate(compressed\_blob) AS VARCHAR)**. See **CPLZLibInflate()** (p. ??).

### 6.1.7 OGR geocoding functions

The following SQL functions are available : **ogr\_geocode(...)** and **ogr\_geocode\_reverse(...)**.

**ogr\_geocode(name\_to\_geocode [, field\_to\_return [, option1 [, option2, ...]])** where **name\_to\_geocode** is a literal or a column name that must be geocoded. **field\_to\_return** if specified can be "geometry" for the geometry (default), or a field name of the layer returned by **OGRGeocode()** (p. ??). The special field "raw" can also be used to return the raw response (XML string) of the geocoding service. **option1**, **option2**, etc.. must be of the key=value format, and are options understood by **OGRGeocodeCreateSession()** (p. ??) or **OGRGeocode()** (p. ??).

This function internally uses the **OGRGeocode()** (p. ??) API. Refer to it for more details.

```
SELECT ST_Centroid(ogr_geocode('Paris'))

returns:

OGRFeature(SELECT):0
  POINT (2.342878767069653 48.85661793020374)

ogrinfo cities.csv -dialect sqlite -sql "SELECT *, ogr_geocode(city, 'country') AS country,
      ST_Centroid(ogr_geocode(city)) FROM cities"

returns:

OGRFeature(SELECT):0
  id (Real) = 1
  city (String) = Paris
  country (String) = France métropolitaine
  POINT (2.342878767069653 48.85661793020374)

OGRFeature(SELECT):1
  id (Real) = 2
  city (String) = London
  country (String) = United Kingdom
  POINT (-0.109369427546499 51.500506667319407)
```

```

OGRFeature(SELECT):2
  id (Real) = 3
  city (String) = Rennes
  country (String) = France métropolitaine
  POINT (-1.68185153381778 48.111663929761093)

OGRFeature(SELECT):3
  id (Real) = 4
  city (String) = Strasbourg
  country (String) = France métropolitaine
  POINT (7.767762859150757 48.571233274141846)

OGRFeature(SELECT):4
  id (Real) = 5
  city (String) = New York
  country (String) = United States of America
  POINT (-73.938140243499049 40.663799577449979)

OGRFeature(SELECT):5
  id (Real) = 6
  city (String) = Berlin
  country (String) = Deutschland
  POINT (13.402306623451983 52.501470321410636)

OGRFeature(SELECT):6
  id (Real) = 7
  city (String) = Beijing
  country (String) =
  POINT (116.391195 39.9064702)

OGRFeature(SELECT):7
  id (Real) = 8
  city (String) = Brasilia
  country (String) = Brasil
  POINT (-52.830435216371839 -10.828214867369699)

OGRFeature(SELECT):8
  id (Real) = 9
  city (String) = Moscow
  country (String) =
  POINT (37.367988106866868 55.556208255649558)

```

**ogr\_geocode\_reverse(longitude, latitude, field\_to\_return [, option1 [, option2, ...]])** where longitude, latitude is the coordinate to query. field\_to\_return must be a field name of the layer returned by **OGRGeocodeReverse()** (p. ??) (for example 'display\_name'). The special field "raw" can also be used to return the raw response (XML string) of the geocoding service. option1, option2, etc.. must be of the key=value format, and are options understood by **OGRGeocodeCreateSession()** (p. ??) or **OGRGeocodeReverse()** (p. ??).

**ogr\_geocode\_reverse(geometry, field\_to\_return [, option1 [, option2, ...]])** is also accepted as an alternate syntax where geometry is a (Spatialite) point geometry.

This function internally uses the **OGRGeocodeReverse()** (p. ??) API. Refer to it for more details.

### 6.1.8 Spatialite spatial index

Spatialite spatial index mechanism can be triggered by making sure a spatial index virtual table is mentioned in the SQL (of the form idx\_layername\_geometrycolumn), or by using the more recent SpatialIndex from the Virtual-SpatialIndex extension. In which case, a in-memory RTree will be built to be used to speed up the spatial queries.

For example, a spatial intersection between 2 layers, by using a spatial index on one of the layers to limit the number of actual geometry intersection computations :

```

SELECT city_name, region_name FROM cities, regions WHERE
  ST_Area(ST_Intersection(cities.geometry, regions.geometry)) > 0 AND
  regions.rowid IN (
    SELECT pkid FROM idx_regions_geometry WHERE
      xmax >= MbrMinX(cities.geometry) AND xmin <= MbrMaxX(cities.geometry) AND
      ymax >= MbrMinY(cities.geometry) AND ymin <= MbrMaxY(cities.geometry))

```

or more elegantly :

```

SELECT city_name, region_name FROM cities, regions WHERE
  ST_Area(ST_Intersection(cities.geometry, regions.geometry)) > 0 AND
  regions.rowid IN (

```

```
SELECT rowid FROM SpatialIndex WHERE  
    f_table_name = 'regions' AND search_frame = cities.geometry)
```



## Chapter 7

# OGR Projections Tutorial

### 7.1 Introduction

The **OGRSpatialReference** (p. ??), and **OGRCoordinateTransformation** (p. ??) classes provide services to represent coordinate systems (projections and datums) and to transform between them. These services are loosely modelled on the OpenGIS Coordinate Transformations specification, and use the same Well Known Text format for describing coordinate systems.

Some background on OpenGIS coordinate systems and services can be found in the Simple Features for COM, and Spatial Reference Systems Abstract Model documents available from the [Open Geospatial Consortium](#). The [GeoTIFF Projections Transform List](#) may also be of assistance in understanding formulations of projections in WKT. The [EPSG Geodesy web page](#) is also a useful resource.

### 7.2 Defining a Geographic Coordinate System

Coordinate systems are encapsulated in the **OGRSpatialReference** (p. ??) class. There are a number of ways of initializing an **OGRSpatialReference** (p. ??) object to a valid coordinate system. There are two primary kinds of coordinate systems. The first is geographic (positions are measured in long/lat) and the second is projected (such as UTM - positions are measured in meters or feet).

A Geographic coordinate system contains information on the datum (which implies an spheroid described by a semi-major axis, and inverse flattening), prime meridian (normally Greenwich), and an angular units type which is normally degrees. The following code initializes a geographic coordinate system on supplying all this information along with a user visible name for the geographic coordinate system.

```
OGRSpatialReference oSRS;  
  
oSRS.SetGeogCS( "My geographic coordinate system",  
               "WGS_1984",  
               "My WGS84 Spheroid",  
               SRS_WGS84_SEMIMAJOR, SRS_WGS84_INVFLATTENING,  
               "Greenwich", 0.0,  
               "degree", SRS_UA_DEGREE_CONV );
```

Of these values, the names "My geographic coordinate system", "My WGS84 Spheroid", "Greenwich" and "degree" are not keys, but are used for display to the user. However, the datum name "WGS\_1984" is used as a key to identify the datum, and there are rules on what values can be used. NOTE: Prepare writeup somewhere on valid datums!

The **OGRSpatialReference** (p. ??) has built in support for a few well known coordinate systems, which include "NAD27", "NAD83", "WGS72" and "WGS84" which can be defined in a single call to `SetWellKnownGeogCS()`.

```
oSRS.SetWellKnownGeogCS( "WGS84" );
```

Furthermore, any geographic coordinate system in the EPSG database can be set by it's GCS code number if the EPSG database is available.

```
oSRS.SetWellKnownGeogCS( "EPSG:4326" );
```

For serialization, and transmission of projection definitions to other packages, the OpenGIS Well Known Text format for coordinate systems is used. An **OGRSpatialReference** (p. ??) can be initialized from well known text, or converted back into well known text.

```
char      *pszWKT = NULL;

oSRS.SetWellKnownGeogCS( "WGS84" );
oSRS.exportToWkt( &pszWKT );
printf( "%s\n", pszWKT );
```

gives something like:

```
GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563,
AUTHORITY["EPSG",7030]],TOWGS84[0,0,0,0,0,0,0],AUTHORITY["EPSG",6326]],
PRIMEM["Greenwich",0,AUTHORITY["EPSG",8901]],UNIT["DMSH",0.0174532925199433,
AUTHORITY["EPSG",9108]],AXIS["Lat",NORTH],AXIS["Long",EAST],AUTHORITY["EPSG",
4326]]
```

or in more readable form:

```
GEOGCS["WGS 84",
  DATUM["WGS_1984",
    SPHEROID["WGS 84",6378137,298.257223563,
      AUTHORITY["EPSG",7030]],
    TOWGS84[0,0,0,0,0,0,0],
    AUTHORITY["EPSG",6326]],
  PRIMEM["Greenwich",0,AUTHORITY["EPSG",8901]],
  UNIT["DMSH",0.0174532925199433,AUTHORITY["EPSG",9108]],
  AXIS["Lat",NORTH],
  AXIS["Long",EAST],
  AUTHORITY["EPSG",4326]]
```

The **OGRSpatialReference::importFromWkt()** (p. ??) method can be used to set an **OGRSpatialReference** (p. ??) from a WKT coordinate system definition.

## 7.3 Defining a Projected Coordinate System

A projected coordinate system (such as UTM, Lambert Conformal Conic, etc) requires an underlying geographic coordinate system as well as a definition for the projection transform used to translate between linear positions (in meters or feet) and angular long/lat positions. The following code defines a UTM zone 17 projected coordinate system with an underlying geographic coordinate system (datum) of WGS84.

```
OGRSpatialReference oSRS;

oSRS.SetProjCS( "UTM 17 (WGS84) in northern hemisphere." );
oSRS.SetWellKnownGeogCS( "WGS84" );
oSRS.SetUTM( 17, TRUE );
```

Calling **SetProjCS()** sets a user name for the projected coordinate system and establishes that the system is projected. The **SetWellKnownGeogCS()** associates a geographic coordinate system, and the **SetUTM()** call sets detailed projection transformation parameters. At this time the above order is important in order to create a valid definition, but in the future the object will automatically reorder the internal representation as needed to remain valid. For now **be careful of the order of steps defining an OGRSpatialReference!**

The above definition would give a WKT version that looks something like the following. Note that the UTM 17 was expanded into the details transverse mercator definition of the UTM zone.

```

PROJCS["UTM 17 (WGS84) in northern hemisphere.",
  GEOGCS["WGS 84",
    DATUM["WGS_1984",
      SPHEROID["WGS 84",6378137,298.257223563,
        AUTHORITY["EPSG",7030]],
      TOWGS84[0,0,0,0,0,0],
      AUTHORITY["EPSG",6326]],
    PRIMEM["Greenwich",0,AUTHORITY["EPSG",8901]],
    UNIT["DMSH",0.0174532925199433,AUTHORITY["EPSG",9108]],
    AXIS["Lat",NORTH],
    AXIS["Long",EAST],
    AUTHORITY["EPSG",4326]],
  PROJECTION["Transverse_Mercator"],
  PARAMETER["latitude_of_origin",0],
  PARAMETER["central_meridian",-81],
  PARAMETER["scale_factor",0.9996],
  PARAMETER["false_easting",500000],
  PARAMETER["false_northing",0]]

```

There are methods for many projection methods including `SetTM()` (Transverse Mercator), `SetLCC()` (Lambert Conformal Conic), and `SetMercator()`.

## 7.4 Querying Coordinate System

Once an **OGRSpatialReference** (p. ??) has been established, various information about it can be queried. It can be established if it is a projected or geographic coordinate system using the **OGRSpatialReference::IsProjected()** (p. ??) and **OGRSpatialReference::IsGeographic()** (p. ??) methods. The **OGRSpatialReference::GetSemiMajor()** (p. ??), **OGRSpatialReference::GetSemiMinor()** (p. ??) and **OGRSpatialReference::GetInvFlattening()** (p. ??) methods can be used to get information about the spheroid. The **OGRSpatialReference::GetAttrValue()** (p. ??) method can be used to get the PROJCS, GEOGCS, DATUM, SPHEROID, and PROJECTION names strings. The **OGRSpatialReference::GetProjParm()** (p. ??) method can be used to get the projection parameters. The **OGRSpatialReference::GetLinearUnits()** (p. ??) method can be used to fetch the linear units type, and translation to meters.

The following code (from `ogr_srs_proj4.cpp`) demonstrates use of `GetAttrValue()` to get the projection, and `GetProjParm()` to get projection parameters. The `GetAttrValue()` method searches for the first "value" node associated with the named entry in the WKT text representation. The `#define'd` constants for projection parameters (such as `SRS_PP_CENTRAL_MERIDIAN`) should be used when fetching projection parameter with `GetProjParm()`. The code for the `Set` methods of the various projections in `ogrsatialreference.cpp` can be consulted to find which parameters apply to which projections.

```

const char *pszProjection = poSRS->GetAttrValue("PROJECTION");

if( pszProjection == NULL )
{
    if( poSRS->IsGeographic() )
        sprintf( szProj4+strlen(szProj4), "+proj=longlat " );
    else
        sprintf( szProj4+strlen(szProj4), "unknown " );
}
else if( EQUAL(pszProjection,SRS_PT_CYLINDRICAL_EQUAL_AREA) )
{
    sprintf( szProj4+strlen(szProj4),
        "+proj=cea +lon_0=%.9f +lat_ts=%.9f +x_0=%.3f +y_0=%.3f ",
        poSRS->GetProjParm(SRS_PP_CENTRAL_MERIDIAN,0.0),
        poSRS->GetProjParm(SRS_PP_STANDARD_PARALLEL_1,0.0),
        poSRS->GetProjParm(SRS_PP_FALSE_EASTING,0.0),
        poSRS->GetProjParm(SRS_PP_FALSE_NORTHING,0.0) );
}
...

```

## 7.5 Coordinate Transformation

The **OGRCoordinateTransformation** (p. ??) class is used for translating positions between different coordinate systems. New transformation objects are created using **OGRCreateCoordinateTransformation()** (p. ??), and then the **OGRCoordinateTransformation::Transform()** (p. ??) method can be used to convert points between coordinate systems.

```
OGRSpatialReference oSourceSRS, oTargetSRS;
OGRCoordinateTransformation *poCT;
double x, y;

oSourceSRS.ImportFromEPSG( atoi(papszArgv[i+1]) );
oTargetSRS.ImportFromEPSG( atoi(papszArgv[i+2]) );

poCT = OGRCreateCoordinateTransformation( &oSourceSRS,
                                          &oTargetSRS );

x = atof( papszArgv[i+3] );
y = atof( papszArgv[i+4] );

if( poCT == NULL || !poCT->Transform( 1, &x, &y ) )
    printf( "Transformation failed.\n" );
else
    printf( "(%f,%f) -> (%f,%f)\n",
            atof( papszArgv[i+3] ),
            atof( papszArgv[i+4] ),
            x, y );
```

There are a couple of points at which transformations can fail. First, **OGRCreateCoordinateTransformation()** (p. ??) may fail, generally because the internals recognise that no transformation between the indicated systems can be established. This might be due to use of a projection not supported by the internal PROJ.4 library, differing datums for which no relationship is known, or one of the coordinate systems being inadequately defined. If **OGRCreateCoordinateTransformation()** (p. ??) fails it will return a NULL.

The **OGRCoordinateTransformation::Transform()** (p. ??) method itself can also fail. This may be as a delayed result of one of the above problems, or as a result of an operation being numerically undefined for one or more of the passed in points. The Transform() function will return TRUE on success, or FALSE if any of the points fail to transform. The point array is left in an indeterminate state on error.

Though not shown above, the coordinate transformation service can take 3D points, and will adjust elevations for elevation differences in spheroids, and datums. At some point in the future shifts between different vertical datums may also be applied. If no Z is passed, it is assumed that the point is on the geoid.

The following example shows how to conveniently create a lat/long coordinate system using the same geographic coordinate system as a projected coordinate system, and using that to transform between projected coordinates and lat/long.

```
OGRSpatialReference oUTM, *poLatLong;
OGRCoordinateTransformation *poTransform;

oUTM.SetProjCS("UTM 17 / WGS84");
oUTM.SetWellKnownGeogCS( "WGS84" );
oUTM.SetUTM( 17 );

poLatLong = oUTM.CloneGeogCS();

poTransform = OGRCreateCoordinateTransformation( &oUTM, poLatLong );
if( poTransform == NULL )
{
    ...
}

...

if( !poTransform->Transform( nPoints, x, y, z ) )
    ...
```

## 7.6 Alternate Interfaces

A C interface to the coordinate system services is defined in **ogr\_srs\_api.h** (p. ??), and Python bindings are available via the **osr.py** module. Methods are close analogs of the C++ methods but C and Python bindings are missing for some C++ methods.



## C Bindings

```

typedef void *OGRSpatialReferenceH;
typedef void *OGRCoordinateTransformationH;

OGRSpatialReferenceH OSRNewSpatialReference( const char * );
void OSRDestroySpatialReference( OGRSpatialReferenceH );

int OSRReference( OGRSpatialReferenceH );
int OSRDereference( OGRSpatialReferenceH );

OGRERR OSRImportFromEPSG( OGRSpatialReferenceH, int );
OGRERR OSRImportFromWkt( OGRSpatialReferenceH, char ** );
OGRERR OSRExportToWkt( OGRSpatialReferenceH, char ** );

OGRERR OSRSetAttrValue( OGRSpatialReferenceH hSRS, const char * pszNodePath,
                        const char * pszNewNodeValue );
const char *OSRGetAttrValue( OGRSpatialReferenceH hSRS,
                             const char * pszName, int iChild);

OGRERR OSRSetLinearUnits( OGRSpatialReferenceH, const char *, double );
double OSRGetLinearUnits( OGRSpatialReferenceH, char ** );

int OSRIsGeographic( OGRSpatialReferenceH );
int OSRIsProjected( OGRSpatialReferenceH );
int OSRIsSameGeogCS( OGRSpatialReferenceH, OGRSpatialReferenceH );
int OSRIsSame( OGRSpatialReferenceH, OGRSpatialReferenceH );

OGRERR OSRSetProjCS( OGRSpatialReferenceH hSRS, const char * pszName );
OGRERR OSRSetWellKnownGeogCS( OGRSpatialReferenceH hSRS,
                              const char * pszName );

OGRERR OSRSetGeogCS( OGRSpatialReferenceH hSRS,
                    const char * pszGeogName,
                    const char * pszDatumName,
                    const char * pszEllipsoidName,
                    double dfSemiMajor, double dfInvFlattening,
                    const char * pszPMName,
                    double dfPMOffset,
                    const char * pszUnits,
                    double dfConvertToRadians );

double OSRGetSemiMajor( OGRSpatialReferenceH, OGRERR * );
double OSRGetSemiMinor( OGRSpatialReferenceH, OGRERR * );
double OSRGetInvFlattening( OGRSpatialReferenceH, OGRERR * );

OGRERR OSRSetAuthority( OGRSpatialReferenceH hSRS,
                      const char * pszTargetKey,
                      const char * pszAuthority,
                      int nCode );
OGRERR OSRSetProjParm( OGRSpatialReferenceH, const char *, double );
double OSRGetProjParm( OGRSpatialReferenceH hSRS,
                      const char * pszParmName,
                      double dfDefault,
                      OGRERR * );

OGRERR OSRSetUTM( OGRSpatialReferenceH hSRS, int nZone, int bNorth );
int OSRGetUTMZone( OGRSpatialReferenceH hSRS, int *pbNorth );

OGRCoordinateTransformationH
OCTNewCoordinateTransformation( OGRSpatialReferenceH hSourceSRS,
                              OGRSpatialReferenceH hTargetSRS );
void OCTDestroyCoordinateTransformation( OGRCoordinateTransformationH );

int OCTTransform( OGRCoordinateTransformationH hCT,
                 int nCount, double *x, double *y, double *z );

```

## Python Bindings

```

class osr.SpatialReference
    def __init__(self, obj=None):
    def ImportFromWkt( self, wkt ):
    def ExportToWkt( self ):
    def ImportFromEPSG( self, code ):
    def IsGeographic( self ):
    def IsProjected( self ):
    def GetAttrValue( self, name, child = 0 ):
    def SetAttrValue( self, name, value ):
    def SetWellKnownGeogCS( self, name ):
    def SetProjCS( self, name = "unnamed" ):
    def IsSameGeogCS( self, other ):
    def IsSame( self, other ):
    def SetLinearUnits( self, units_name, to_meters ):
    def SetUTM( self, zone, is_north = 1 ):

```

```
class CoordinateTransformation:
    def __init__(self, source, target):
    def TransformPoint(self, x, y, z = 0):
    def TransformPoints(self, points):
```

## 7.7 Internal Implementation

The **OGRCoordinateTransformation** (p. ??) service is implemented on top of the PROJ.4 library originally written by Gerald Evenden of the USGS.

## Chapter 8

# Deprecated List

**Member OGR\_G\_GetArea (p. ??) (OGRGeometryH) CPL\_WARN\_DEPRECATED("Non standard method. Use OGR\_G\_Area() instead")**

**Member OGR\_G\_GetBoundary (p. ??) (OGRGeometryH) CPL\_WARN\_DEPRECATED("Non standard method. Use OGR\_G\_Boundary() instead")**

**Member OGR\_G\_SymmetricDifference (p. ??) (OGRGeometryH, OGRGeometryH) CPL\_WARN\_DEPRECATED("Non standard method. Use OGR\_G\_SymDifference() instead")**

**Member OGRGeometry::getBoundary (p. ??) () const CPL\_WARN\_DEPRECATED("Non standard method. Use Boundary() instead")**

**Member OGRGeometry::SymmetricDifference (p. ??) (const OGRGeometry (p. ??) \*) const CPL\_WARN\_DEPRECATED("Non standard method. Use SymDifference() instead")**

**Member OGRLayer::GetInfo (p. ??) (const char \*)**

**Member OGRSpatialReference::importFromOzi (p. ??) (const char \*, const char \*, const char \*)**

Use importFromOzi( const char \* const\* papszLines ) instead

**Member OGRSpatialReference::~~OGRSpatialReference (p. ??) ()**



## Chapter 9

# Hierarchical Index

### 9.1 Class Hierarchy

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

_CPLHashSet . . . . .	??
_CPLList . . . . .	??
_CPLQuadTree . . . . .	??
_CPLSpawnedProcess . . . . .	??
_MutexLinkedElt . . . . .	??
_OGRGeocodingSessionHS . . . . .	??
_QuadTreeNode . . . . .	??
_sPolyExtended . . . . .	??
CachedConnection . . . . .	??
CachedDirList . . . . .	??
CachedFileProp . . . . .	??
CachedRegion . . . . .	??
CPLErrorContext . . . . .	??
CPLHTTPResult . . . . .	??
CPLKeywordParser . . . . .	??
CPLLocaleC . . . . .	??
CPLMimePart . . . . .	??
CPLMutexHolder . . . . .	??
CPLODBCDriverInstaller . . . . .	??
CPLODBCSession . . . . .	??
CPLODBCStatement . . . . .	??
CPLRectObj . . . . .	??
CPLSharedFileInfo . . . . .	??
CPLSharedFileInfoExtra . . . . .	??
CPLStdCallThreadInfo . . . . .	??
CPLStringList . . . . .	??
CPLXMLNode . . . . .	??
CPLZip . . . . .	??
ctb . . . . .	??
curfile_info . . . . .	??
DefaultCSVFileNameTLS . . . . .	??
errHandler . . . . .	??
file_in_zip_read_info_s . . . . .	??
FindFileTLS . . . . .	??
GDALScaledProgressInfo . . . . .	??
GZipSnapshot . . . . .	??
linkedlist_data_s . . . . .	??
linkedlist_datablock_internal_s . . . . .	??

OGR_SRSNode . . . . .	??
ogr_style_param . . . . .	??
ogr_style_value . . . . .	??
OGRAttrIndex . . . . .	??
OGRMIAAttrIndex . . . . .	??
OGRCoordinateTransformation . . . . .	??
OGRProj4CT . . . . .	??
OGRDataSource . . . . .	??
OGREnvelope . . . . .	??
OGREnvelope3D . . . . .	??
OGRFeature . . . . .	??
OGRFeatureDefn . . . . .	??
OGRFeatureQuery . . . . .	??
OGRField . . . . .	??
OGRFieldDefn . . . . .	??
OGRGeometry . . . . .	??
OGRCurve . . . . .	??
OGRLineString . . . . .	??
OGRLinearRing . . . . .	??
OGRGeometryCollection . . . . .	??
OGRMultiLineString . . . . .	??
OGRMultiPoint . . . . .	??
OGRMultiPolygon . . . . .	??
OGRPoint . . . . .	??
OGRSurface . . . . .	??
OGRPolygon . . . . .	??
OGRGeometryFactory . . . . .	??
OGRLayer . . . . .	??
OGRAbstractProxiedLayer . . . . .	??
OGRProxiedLayer . . . . .	??
OGRGenSQLResultsLayer . . . . .	??
OGRLayerDecorator . . . . .	??
OGRWarpedLayer . . . . .	??
OGRUnionLayer . . . . .	??
OGRLayerAttrIndex . . . . .	??
OGRMILayerAttrIndex . . . . .	??
OGRLayerPool . . . . .	??
OGRProj4Datum . . . . .	??
OGRProj4PM . . . . .	??
OGRRawPoint . . . . .	??
OGRSFDriver . . . . .	??
OGRSFDriverRegistrar . . . . .	??
OGRSpatialReference . . . . .	??
OGRStyleMgr . . . . .	??
OGRStyleTable . . . . .	??
OGRStyleTool . . . . .	??
OGRStyleBrush . . . . .	??
OGRStyleLabel . . . . .	??
OGRStylePen . . . . .	??
OGRStyleSymbol . . . . .	??
ParseContext . . . . .	??
PCIDatums . . . . .	??
projUV . . . . .	??
RingBuffer . . . . .	??
SFRegion . . . . .	??
StackContext . . . . .	??

string	
CPLString	??
swq_col_def	??
swq_expr_node	??
swq_field_list	??
swq_join_def	??
swq_op_registrar	??
swq_operation	??
swq_order_def	??
swq_parse_context	??
swq_select	??
swq_summary	??
swq_table_def	??
tm_unz_s	??
tm_zip_s	??
unz_file_info_internal_s	??
unz_file_info_s	??
unz_file_pos_s	??
unz_global_info_s	??
unz_s	??
VSIArchiveContent	??
VSIArchiveEntry	??
VSIArchiveEntryFileOffset	??
VSITarEntryFileOffset	??
VSIZipEntryFileOffset	??
VSIArchiveReader	??
VSITarReader	??
VSIZipReader	??
VSIChunk	??
VSIFileManager	??
VSIFilesystemHandler	??
VSIArchiveFilesystemHandler	??
VSITarFilesystemHandler	??
VSIZipFilesystemHandler	??
VSIcurlFilesystemHandler	??
VSIcurlStreamingFSHandler	??
VSIgzipFilesystemHandler	??
VSImemoryFilesystemHandler	??
VSIsparseFileFilesystemHandler	??
VSIstdinFilesystemHandler	??
VSIstdoutFilesystemHandler	??
VSIstdoutRedirectFilesystemHandler	??
VSIsubFileFilesystemHandler	??
VSIunixStdioFilesystemHandler	??
VSIMemFile	??
VSIReadDirRecursiveTask	??
VSIVirtualHandle	??
VSIBufferedReaderHandle	??
VSICachedFile	??
VSIcurlHandle	??
VSIcurlStreamingHandle	??
VSIgzipHandle	??
VSIgzipWriteHandle	??
VSImemoryHandle	??
VSIsparseFileHandle	??
VSIstdinHandle	??
VSIstdoutHandle	??

VSIStdoutRedirectHandle . . . . .	??
VSISubFileHandle . . . . .	??
VSIUnixStdioHandle . . . . .	??
VSIZipWriteHandle . . . . .	??
WriteFuncStruct . . . . .	??
yyalloc . . . . .	??
zip_fileinfo . . . . .	??
zip_internal . . . . .	??
zlib_filefunc_def_s . . . . .	??



## Chapter 10

# Class Index

### 10.1 Class List

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

<b>_CPLHashSet</b>	??
<b>_CPLList</b>	??
<b>_CPLQuadTree</b>	??
<b>_CPLSpawnedProcess</b>	??
<b>_MutexLinkedElt</b>	??
<b>_OGRGeocodingSessionHS</b>	??
<b>_QuadTreeNode</b>	??
<b>_sPolyExtended</b>	??
<b>CachedConnection</b>	??
<b>CachedDirList</b>	??
<b>CachedFileProp</b>	??
<b>CachedRegion</b>	??
<b>CPLErrorContext</b>	??
<b>CPLHTTPResult</b>	??
<b>CPLKeywordParser</b>	??
<b>CPLLocaleC</b>	??
<b>CPLMimePart</b>	??
<b>CPLMutexHolder</b>	??
<b>CPLODBCDriverInstaller</b>	??
<b>CPLODBCSession</b>	??
<b>CPLODBCStatement</b>	??
<b>CPLRectObj</b>	??
<b>CPLSharedFileInfo</b>	??
<b>CPLSharedFileInfoExtra</b>	??
<b>CPLStdCallThreadInfo</b>	??
<b>CPLString</b>	
Convenient string class based on std::string	??
<b>CPLStringList</b>	
String list class designed around our use of C "char**" string lists	??
<b>CPLXMLNode</b>	??
<b>CPLZip</b>	??
<b>ctb</b>	??
<b>curfile_info</b>	??
<b>DefaultCSVFileNameTLS</b>	??
<b>errHandler</b>	??
<b>file_in_zip_read_info_s</b>	??
<b>FindFileTLS</b>	??
<b>GDALScaledProgressInfo</b>	??

GZipSnapshot	??
linkedlist_data_s	??
linkedlist_datablock_internal_s	??
OGR_SRSNode	??
ogr_style_param	??
ogr_style_value	??
OGRAbstractProxiedLayer	??
OGRAttrIndex	??
OGRCoordinateTransformation	??
OGRCurve	??
OGRDataSource	??
OGREnvelope	??
OGREnvelope3D	??
OGRFeature	??
OGRFeatureDefn	??
OGRFeatureQuery	??
OGRField	??
OGRFieldDefn	??
OGRGenSQLResultsLayer	??
OGRGeometry	??
OGRGeometryCollection	??
OGRGeometryFactory	??
OGRLayer	??
OGRLayerAttrIndex	??
OGRLayerDecorator	??
OGRLayerPool	??
OGRLinearRing	??
OGRLineString	??
OGRMIAAttrIndex	??
OGRMILayerAttrIndex	??
OGRMultiLineString	??
OGRMultiPoint	??
OGRMultiPolygon	??
OGRPoint	??
OGRPolygon	??
OGRProj4CT	??
OGRProj4Datum	??
OGRProj4PM	??
OGRProxiedLayer	??
OGRRawPoint	??
OGRSFDriver	??
OGRSFDriverRegistrar	??
OGRSpatialReference	??
OGRStyleBrush	??
OGRStyleLabel	??
OGRStyleMgr	??
OGRStylePen	??
OGRStyleSymbol	??
OGRStyleTable	??
OGRStyleTool	??
OGRSurface	??
OGRUnionLayer	??
OGRWarpedLayer	??
ParseContext	??
PCIDatums	??
projUV	??
RingBuffer	??
SFRegion	??

StackContext	??
swq_col_def	??
swq_expr_node	??
swq_field_list	??
swq_join_def	??
swq_op_registrar	??
swq_operation	??
swq_order_def	??
swq_parse_context	??
swq_select	??
swq_summary	??
swq_table_def	??
tm_unz_s	??
tm_zip_s	??
unz_file_info_internal_s	??
unz_file_info_s	??
unz_file_pos_s	??
unz_global_info_s	??
unz_s	??
VSIArchiveContent	??
VSIArchiveEntry	??
VSIArchiveEntryFileOffset	??
VSIArchiveFilesystemHandler	??
VSIArchiveReader	??
VSIBufferedReaderHandle	??
VSIChunk	??
VSICachedFile	??
VSIcurlFilesystemHandler	??
VSIcurlHandle	??
VSIcurlStreamingFSHandler	??
VSIcurlStreamingHandle	??
VSIFileManager	??
VSIFilesystemHandler	??
VSIGZipFilesystemHandler	??
VSIGZipHandle	??
VSIGZipWriteHandle	??
VSIMemFile	??
VSIMemFilesystemHandler	??
VSIMemHandle	??
VSIReadDirRecursiveTask	??
VSIsparseFileFilesystemHandler	??
VSIsparseFileHandle	??
VSIstdinFilesystemHandler	??
VSIstdinHandle	??
VSIstdoutFilesystemHandler	??
VSIstdoutHandle	??
VSIstdoutRedirectFilesystemHandler	??
VSIstdoutRedirectHandle	??
VSIsubFileFilesystemHandler	??
VSIsubFileHandle	??
VSItarEntryFileOffset	??
VSItarFilesystemHandler	??
VSItarReader	??
VSIUnixStdioFilesystemHandler	??
VSIUnixStdioHandle	??
VSVirtualHandle	??
VSIZipEntryFileOffset	??
VSIZipFilesystemHandler	??

---

<b>VSZipReader</b>	??
<b>VSZipWriteHandle</b>	??
<b>WriteFuncStruct</b>	??
<b>yyalloc</b>	??
<b>zip_fileinfo</b>	??
<b>zip_internal</b>	??
<b>zlib_filefunc_def_s</b>	??

# Chapter 11

## File Index

### 11.1 File List

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

cpl_atomic_ops.h	??
cpl_config.h	??
cpl_config_extras.h	??
cpl_conv.h	??
cpl_csv.h	??
cpl_error.h	??
cpl_hash_set.h	??
cpl_http.h	??
cpl_list.h	??
cpl_minixml.h	??
cpl_minizip_ioapi.h	??
cpl_minizip_unzip.h	??
cpl_minizip_zip.h	??
cpl_multiproc.h	??
cpl_odbc.h	??
cpl_port.h	??
cpl_progress.h	??
cpl_quad_tree.h	??
cpl_spawn.h	??
cpl_string.h	??
cpl_time.h	??
cpl_vsi.h	??
cpl_vsi_virtual.h	??
cpl_vsil_curl_priv.h	??
cpl_win32ce_api.h	??
cpl_wince.h	??
cplkeywordparser.h	??
gdal_csv.h	??
ogr_api.h	??
ogr_attrind.h	??
ogr_core.h	??
ogr_expat.h	??
ogr_feature.h	??
ogr_featurestyle.h	??
ogr_gensql.h	??
ogr_geocoding.h	??
ogr_geometry.h	??
ogr_geos.h	??

ogr_p.h	??
ogr_spatialref.h	??
ogr_srs_api.h	??
ogr_srs_esri_names.h	??
ogrgeomediageometry.h	??
ogrlayerdecorator.h	??
ogrlayerpool.h	??
ogrpgeogeometry.h	??
ogrsf_frmts.h	??
ogrunionlayer.h	??
ogrwarpedlayer.h	??
swq.h	??

# Chapter 12

## Class Documentation

### 12.1 `_CPLHashSet` Struct Reference

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

- `cpl_hash_set.cpp`

### 12.2 `_CPLList` Struct Reference

```
#include <cpl_list.h>
```

#### Public Attributes

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

#### 12.2.1 Detailed Description

List element structure.

#### 12.2.2 Member Data Documentation

##### 12.2.2.1 `void* _CPLList::pData`

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

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

##### 12.2.2.2 `struct _CPLList* _CPLList::psNext`

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

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

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

- `cpl_list.h`

## 12.3 `_CPLQuadTree` Struct Reference

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

- `cpl_quad_tree.cpp`

## 12.4 `_CPLSpawnedProcess` Struct Reference

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

- `cpl_spawn.cpp`

## 12.5 `_MutexLinkedElt` Struct Reference

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

- `cpl_multiproc.cpp`

## 12.6 `_OGRGeocodingSessionHS` Struct Reference

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

- `ogr_geocoding.cpp`

## 12.7 `_QuadTreeNode` Struct Reference

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

- `cpl_quad_tree.cpp`

## 12.8 `_sPolyExtended` Struct Reference

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

- `ogrgeometryfactory.cpp`

## 12.9 `CachedConnection` Struct Reference

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

- `cpl_vsil_curl.cpp`



## 12.10 CachedDirList Struct Reference

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

- `cpl_vsil_curl.cpp`

## 12.11 CachedFileProp Struct Reference

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

- `cpl_vsil_curl.cpp`
- `cpl_vsil_curl_streaming.cpp`

## 12.12 CachedRegion Struct Reference

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

- `cpl_vsil_curl.cpp`

## 12.13 CPLErrorContext Struct Reference

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

- `cpl_error.cpp`

## 12.14 CPLHTTPResult Struct Reference

```
#include <cpl_http.h>
```

### Public Attributes

- `int nStatus`
- `char * pszContentType`
- `char * pszErrBuf`
- `int nDataLen`
- `GByte * pabyData`
- `char ** papszHeaders`
- `int nMimePartCount`
- `CPLMimePart * pasMimePart`

### 12.14.1 Detailed Description

Describe the result of a `CPLHTTPFetch()` (p. ??) call

## 12.14.2 Member Data Documentation

### 12.14.2.1 `int CPLHTTPResult::nDataLen`

Length of the pabyData buffer

Referenced by `CPLHTTPFetch()`, `CPLHTTPParseMultipartMime()`, and `OGRSpatialReference::importFromUrl()`.

### 12.14.2.2 `int CPLHTTPResult::nMimePartCount`

Number of parts in a multipart message

Referenced by `CPLHTTPDestroyResult()`, and `CPLHTTPParseMultipartMime()`.

### 12.14.2.3 `int CPLHTTPResult::nStatus`

cURL error code : 0=success, non-zero if request failed

Referenced by `CPLHTTPFetch()`, and `OGRSpatialReference::importFromUrl()`.

### 12.14.2.4 `GByte* CPLHTTPResult::pabyData`

Buffer with downloaded data

Referenced by `CPLHTTPDestroyResult()`, `CPLHTTPParseMultipartMime()`, `GOA2GetAccessToken()`, `GOA2GetRefreshToken()`, and `OGRSpatialReference::importFromUrl()`.

### 12.14.2.5 `char** CPLHTTPResult::papszHeaders`

Headers returned

Referenced by `CPLHTTPDestroyResult()`, and `CPLHTTPFetch()`.

### 12.14.2.6 `CPLMimePart* CPLHTTPResult::pasMimePart`

Array of parts (resolved by `CPLHTTPParseMultipartMime()` (p. ??))

Referenced by `CPLHTTPDestroyResult()`, and `CPLHTTPParseMultipartMime()`.

### 12.14.2.7 `char* CPLHTTPResult::pszContentType`

Content-Type of the response

Referenced by `CPLHTTPDestroyResult()`, `CPLHTTPFetch()`, and `CPLHTTPParseMultipartMime()`.

### 12.14.2.8 `char* CPLHTTPResult::pszErrMsgBuf`

Error message from curl, or NULL

Referenced by `CPLHTTPDestroyResult()`, `CPLHTTPFetch()`, `GOA2GetAccessToken()`, `GOA2GetRefreshToken()`, and `OGRSpatialReference::importFromUrl()`.

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

- `cpl_http.h`

## 12.15 CPLKeywordParser Class Reference

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

- `cplkeywordparser.h`
- `cplkeywordparser.cpp`

## 12.16 CPLLocaleC Class Reference

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

- `cpl_conv.h`
- `cpl_conv.cpp`

## 12.17 CPLMimePart Struct Reference

```
#include <cpl_http.h>
```

### Public Attributes

- `char ** papszHeaders`
- `GByte * pabyData`
- `int nDataLen`

### 12.17.1 Detailed Description

Describe a part of a multipart message

### 12.17.2 Member Data Documentation

#### 12.17.2.1 `int CPLMimePart::nDataLen`

Buffer length

Referenced by `CPLHTTPParseMultipartMime()`.

#### 12.17.2.2 `GByte* CPLMimePart::pabyData`

Buffer with data of the part

Referenced by `CPLHTTPParseMultipartMime()`.

#### 12.17.2.3 `char** CPLMimePart::papszHeaders`

NULL terminated array of headers

Referenced by `CPLHTTPDestroyResult()`, and `CPLHTTPParseMultipartMime()`.

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

- `cpl_http.h`

## 12.18 CPLMutexHolder Class Reference

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

- cpl\_multiproc.h
- cpl\_multiproc.cpp

## 12.19 CPODBCDriverInstaller Class Reference

```
#include <cpl_odbc.h>
```

### Public Member Functions

- int **InstallDriver** (const char \*pszDriver, const char \*pszPathIn, WORD fRequest=ODBC\_INSTALL\_COMPLETE)
- int **RemoveDriver** (const char \*pszDriverName, int fRemoveDSN=0)

### 12.19.1 Detailed Description

A class providing functions to install or remove ODBC driver.

### 12.19.2 Member Function Documentation

**12.19.2.1** int CPODBCDriverInstaller::InstallDriver ( const char \* *pszDriver*, const char \* *pszPathIn*, WORD *fRequest* = ODBC\_INSTALL\_COMPLETE )

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

#### Parameters

<i>pszDriver</i>	- The driver definition as a list of keyword-value pairs describing the driver (See ODBC API Reference).
<i>pszPathIn</i>	- Full path of the target directory of the installation, or a null pointer (for unixODBC, NULL is passed).
<i>fRequest</i>	- The fRequest argument must contain one of the following values: ODBC_INSTALL_COMPLETE - (default) complete the installation request ODBC_INSTALL_INQUIRY - inquire about where a driver can be installed

#### Returns

TRUE indicates success, FALSE if it fails.

References CPLDebug(), and CPLMalloc().

**12.19.2.2** int CPODBCDriverInstaller::RemoveDriver ( const char \* *pszDriverName*, int *fRemoveDSN* = 0 )

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

## Parameters

<i>pszDriverName</i>	- The name of the driver as registered in the Odbcinst.ini key of the system information.
<i>fRemoveDSN</i>	- TRUE: Remove DSNs associated with the driver specified in <i>lpszDriver</i> . FALSE: Do not remove DSNs associated with the driver specified in <i>lpszDriver</i> .

## Returns

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

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

- `cpl_odbc.h`
- `cpl_odbc.cpp`

## 12.20 CPODBCSession Class Reference

```
#include <cpl_odbc.h>
```

### Public Member Functions

- `int EstablishSession` (const char \**pszDSN*, const char \**pszUserid*, const char \**pszPassword*)
- `const char * GetLastError` ()

#### 12.20.1 Detailed Description

A class representing an ODBC database session.

Includes error collection services.

#### 12.20.2 Member Function Documentation

**12.20.2.1** `int CPODBCSession::EstablishSession ( const char * pszDSN, const char * pszUserid, const char * pszPassword )`

Connect to database and logon.

## Parameters

<i>pszDSN</i>	The name of the DSN being used to connect. This is not optional.
<i>pszUserid</i>	the userid to logon as, may be NULL if not required, or provided by the DSN.
<i>pszPassword</i>	the password to logon with. May be NULL if not required or provided by the DSN.

## Returns

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

References `CPLDebug()`, and `GetLastError()`.

**12.20.2.2** `const char * CPODBCSession::GetLastError ( )`

Returns the last ODBC error message.

**Returns**

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

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

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

- **cpl\_odbc.h**
- **cpl\_odbc.cpp**

## 12.21 CPLODBCStatement Class Reference

```
#include <cpl_odbc.h>
```

### Public Member Functions

- void **Clear** ()
- void **AppendEscaped** (const char \*)
- void **Append** (const char \*)
- void **Append** (int)
- void **Append** (double)
- int **Appendf** (const char \*,...)
- int **ExecuteSQL** (const char \*=0)
- int **Fetch** (int nOrientation=SQL\_FETCH\_NEXT, int nOffset=0)
- int **GetColCount** ()
- const char \* **GetColName** (int)
- short **GetColType** (int)
- const char \* **GetColTypeName** (int)
- short **GetColSize** (int)
- short **GetColPrecision** (int)
- short **GetColNullable** (int)
- int **GetColId** (const char \*)
- const char \* **GetColData** (int, const char \*=0)
- const char \* **GetColData** (const char \*, const char \*=0)
- int **GetColumns** (const char \*pszTable, const char \*pszCatalog=0, const char \*pszSchema=0)
- int **GetPrimaryKeys** (const char \*pszTable, const char \*pszCatalog=0, const char \*pszSchema=0)
- int **GetTables** (const char \*pszCatalog=0, const char \*pszSchema=0)
- void **DumpResult** (FILE \*fp, int bShowSchema=0)

### Static Public Member Functions

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

#### 12.21.1 Detailed Description

Abstraction for statement, and resultset.

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

### 12.21.2 Member Function Documentation

#### 12.21.2.1 void CPLODBCStatement::Append ( const char \* *pszText* )

Append text to internal command.

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

Parameters

<i>pszText</i>	text to append.
----------------	-----------------

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

#### 12.21.2.2 void CPLODBCStatement::Append ( int *nValue* )

Append to internal command.

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

Parameters

<i>nValue</i>	value to append to the command.
---------------	---------------------------------

References Append().

#### 12.21.2.3 void CPLODBCStatement::Append ( double *dfValue* )

Append to internal command.

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

Parameters

<i>dfValue</i>	value to append to the command.
----------------	---------------------------------

References Append().

#### 12.21.2.4 void CPLODBCStatement::AppendEscaped ( const char \* *pszText* )

Append text to internal command.

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

Parameters

<i>pszText</i>	text to append.
----------------	-----------------

References Append().

#### 12.21.2.5 int CPLODBCStatement::Appendf ( const char \* *pszFormat*, ... )

Append to internal command.

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

Parameters

---

<i>pszFormat</i>	printf() style format string.
------------------	-------------------------------

**Returns**

FALSE if formatting fails due to result being too large.

References Append().

**12.21.2.6 void CPODBCStatement::Clear ( )**

Clear internal command text and result set definitions.

References CSLDestroy().

Referenced by ExecuteSQL().

**12.21.2.7 void CPODBCStatement::DumpResult ( FILE \* fp, int bShowSchema = 0 )**

Dump resultset to file.

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

**Parameters**

<i>fp</i>	the file to write to. stdout or stderr are acceptable.
<i>bShowSchema</i>	TRUE to force writing schema information for the rowset before the rowset data itself. Default is FALSE.

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

**12.21.2.8 int CPODBCStatement::ExecuteSQL ( const char \* pszStatement = 0 )**

Execute an SQL statement.

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

**Parameters**

<i>pszStatement</i>	the SQL statement to execute, or NULL if the internally saved one should be used.
---------------------	---

**Returns**

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

References Append(), and Clear().

**12.21.2.9 int CPODBCStatement::Fetch ( int nOrientation = SQL\_FETCH\_NEXT, int nOffset = 0 )**

Fetch a new record.

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

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



## Parameters

<i>nOrientation</i>	One of SQL_FETCH_NEXT, SQL_FETCH_LAST, SQL_FETCH_PRIOR, SQL_FETCH_ABSOLUTE, or SQL_FETCH_RELATIVE (default is SQL_FETCH_NEXT).
<i>nOffset</i>	the offset (number of records), ignored for some orientations.

## Returns

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

References CPLError(), CPLMalloc(), CPLRealloc(), CPLRecodeFromWChar(), CPLODBCSession::GetLastError(), and GetTypeMapping().

Referenced by DumpResult().

## 12.21.2.10 int CPLODBCStatement::GetColCount ( )

Fetch the resultset column count.

## Returns

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

Referenced by DumpResult().

## 12.21.2.11 const char \* CPLODBCStatement::GetColData ( int iCol, const char \* pszDefault = 0 )

Fetch column data.

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

## Parameters

<i>iCol</i>	the zero based column to fetch.
<i>pszDefault</i>	the value to return if the column does not exist, or is NULL. Defaults to NULL.

## Returns

pointer to internal column data or NULL on failure.

Referenced by DumpResult(), and GetColData().

## 12.21.2.12 const char \* CPLODBCStatement::GetColData ( const char \* pszColName, const char \* pszDefault = 0 )

Fetch column data.

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

## Parameters

<i>pszColName</i>	the name of the column requested.
<i>pszDefault</i>	the value to return if the column does not exist, or is NULL. Defaults to NULL.

## Returns

pointer to internal column data or NULL on failure.

References GetColData(), and GetColId().

**12.21.2.13** `int CPODBCStatement::GetColId ( const char * pszColName )`

Fetch column index.

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

**Parameters**

<i>pszColName</i>	the name to search for.
-------------------	-------------------------

**Returns**

the column index, or -1 if not found.

Referenced by GetColData().

**12.21.2.14** `const char * CPODBCStatement::GetColName ( int iCol )`

Fetch a column name.

**Parameters**

<i>iCol</i>	the zero based column index.
-------------	------------------------------

**Returns**

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

Referenced by DumpResult().

**12.21.2.15** `short CPODBCStatement::GetColNullable ( int iCol )`

Fetch the column nullability.

**Parameters**

<i>iCol</i>	the zero based column index.
-------------	------------------------------

**Returns**

TRUE if the column may contains or FALSE otherwise.

Referenced by DumpResult().

**12.21.2.16** `short CPODBCStatement::GetColPrecision ( int iCol )`

Fetch the column precision.

**Parameters**

<i>iCol</i>	the zero based column index.
-------------	------------------------------

**Returns**

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

Referenced by DumpResult().

**12.21.2.17** `short CPODBCStatement::GetColSize ( int iCol )`

Fetch the column width.

## Parameters

<i>iCol</i>	the zero based column index.
-------------	------------------------------

## Returns

column width, zero for unknown width columns.

Referenced by DumpResult().

12.21.2.18 short CPODBCStatement::GetColType ( int *iCol* )

Fetch a column data type.

The return type code is a an ODBC SQL\_ code, one of SQL\_UNKNOWN\_TYPE, SQL\_CHAR, SQL\_NUMERIC, SQL\_DECIMAL, SQL\_INTEGER, SQL\_SMALLINT, SQL\_FLOAT, SQL\_REAL, SQL\_DOUBLE, SQL\_DATETIME, SQL\_VARCHAR, SQL\_TYPE\_DATE, SQL\_TYPE\_TIME, SQL\_TYPE\_TIMESTAMP.

## Parameters

<i>iCol</i>	the zero based column index.
-------------	------------------------------

## Returns

type code or -1 if the column is illegal.

Referenced by DumpResult().

12.21.2.19 const char \* CPODBCStatement::GetColTypeName ( int *iCol* )

Fetch a column data type name.

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

## Parameters

<i>iCol</i>	the zero based column index.
-------------	------------------------------

## Returns

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

12.21.2.20 int CPODBCStatement::GetColumns ( const char \* *pszTable*, const char \* *pszCatalog* = 0, const char \* *pszSchema* = 0 )

Fetch column definitions for a table.

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

## Parameters

<i>pszTable</i>	the name of the table to query information on. This should not be empty.
-----------------	--

<i>pszCatalog</i>	the catalog to find the table in, use NULL (the default) if no catalog is available.
<i>pszSchema</i>	the schema to find the table in, use NULL (the default) if no schema is available.

**Returns**

TRUE on success or FALSE on failure.

References CPLCalloc(), and CPLStrdup().

**12.21.2.21** `int CPODBCStatement::GetPrimaryKeys ( const char * pszTable, const char * pszCatalog = 0, const char * pszSchema = 0 )`

Fetch primary keys for a table.

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

**Parameters**

<i>pszTable</i>	the name of the table to query information on. This should not be empty.
<i>pszCatalog</i>	the catalog to find the table in, use NULL (the default) if no catalog is available.
<i>pszSchema</i>	the schema to find the table in, use NULL (the default) if no schema is available.

**Returns**

TRUE on success or FALSE on failure.

**12.21.2.22** `int CPODBCStatement::GetTables ( const char * pszCatalog = 0, const char * pszSchema = 0 )`

Fetch tables in database.

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

**Parameters**

<i>pszCatalog</i>	the catalog to find the table in, use NULL (the default) if no catalog is available.
<i>pszSchema</i>	the schema to find the table in, use NULL (the default) if no schema is available.

**Returns**

TRUE on success or FALSE on failure.

References CPLDebug().

**12.21.2.23** `SQLSMALLINT CPODBCStatement::GetTypeMapping ( SQLSMALLINT nTypeCode ) [static]`

Get appropriate C data type for SQL column type.

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

## Parameters

<i>nTypeCode</i>	the SQL_ code, such as SQL_CHAR.
------------------	----------------------------------

## Returns

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

Referenced by Fetch().

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

Get name for SQL column type.

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

## Parameters

<i>nTypeCode</i>	the SQL_ code, such as SQL_CHAR.
------------------	----------------------------------

## Returns

internal string, "UNKNOWN" if code not recognised.

Referenced by DumpResult().

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

- **cpl\_odbc.h**
- cpl\_odbc.cpp

## 12.22 CPLRectObj Struct Reference

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

- **cpl\_quad\_tree.h**

## 12.23 CPLSharedFileInfo Struct Reference

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

- **cpl\_conv.h**

## 12.24 CPLSharedFileInfoExtra Struct Reference

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

- cpl\_conv.cpp

## 12.25 CPLStdCallThreadInfo Struct Reference

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

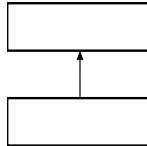
- cpl\_multiproc.cpp

## 12.26 CPLString Class Reference

Convenient string class based on `std::string`.

```
#include <cpl_string.h>
```

Inheritance diagram for CPLString:



### Public Member Functions

- **CPLString & FormatC** (double *dfValue*, const char \**pszFormat*=0)
- **CPLString & Trim** ()
- **size\_t ifind** (const std::string &*str*, size\_t *pos*=0) const
- **size\_t ifind** (const char \**s*, size\_t *pos*=0) const
- **CPLString & toupper** (void)
- **CPLString & tolower** (void)

### 12.26.1 Detailed Description

Convenient string class based on `std::string`.

### 12.26.2 Member Function Documentation

#### 12.26.2.1 CPLString & CPLString::FormatC ( double *dfValue*, const char \* *pszFormat* = 0 )

Format double in C locale.

The passed value is formatted using the C locale (period as decimal separator) and appended to the target **CPLString** (p. ??).

Parameters

<i>dfValue</i>	the value to format.
<i>pszFormat</i>	the sprintf() style format to use or omit for default. Note that this format string should only include one substitution argument and it must be for a double (f or g).

Returns

a reference to the **CPLString** (p. ??).

#### 12.26.2.2 size\_t CPLString::ifind ( const std::string & *str*, size\_t *pos* = 0 ) const

Case insensitive find() alternative.

Parameters

<i>str</i>	substring to find.
<i>pos</i>	offset in the string at which the search starts.

**Returns**

the position of substring in the string or `std::string::npos` if not found.

**Since**

GDAL 1.9.0

Referenced by `CPLURLAddKVP()`, and `CPLURLGetValue()`.

**12.26.2.3 `size_t CPLString::ifind ( const char * s, size_t nPos = 0 ) const`**

Case insensitive `find()` alternative.

**Parameters**

<i>s</i>	substring to find.
<i>nPos</i>	offset in the string at which the search starts.

**Returns**

the position of the substring in the string or `std::string::npos` if not found.

**Since**

GDAL 1.9.0

References `tolower()`.

**12.26.2.4 `CPLString & CPLString::tolower ( void )`**

Convert to lower case in place.

Referenced by `ifind()`.

**12.26.2.5 `CPLString & CPLString::toupper ( void )`**

Convert to upper case in place.

**12.26.2.6 `CPLString & CPLString::Trim ( )`**

Trim white space.

Trims white space off the left and right of the string. White space is any of a space, a tab, a newline ( ' ' ) or a carriage control ( " " ).

**Returns**

a reference to the **CPLString** (p. ??).

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

- `cpl_string.h`
- `cplstring.cpp`

## 12.27 CPLStringList Class Reference

String list class designed around our use of C "char\*" string lists.

```
#include <cpl_string.h>
```

### Public Member Functions

- **CPLStringList** (char \*\*papszList, int bTakeOwnership=1)
- **CPLStringList** (const **CPLStringList** &oOther)  
*Copy constructor.*
- **CPLStringList & Clear** ()
- int **Count** () const
- **CPLStringList & AddString** (const char \*pszNewString)
- **CPLStringList & AddStringDirectly** (char \*pszNewString)
- **CPLStringList & InsertString** (int nInsertAtLineNo, const char \*pszNewLine)  
*Insert into the list at identified location.*
- **CPLStringList & InsertStringDirectly** (int nInsertAtLineNo, char \*pszNewLine)
- int **FindName** (const char \*pszName) const
- int **FetchBoolean** (const char \*pszKey, int bDefault) const
- const char \* **FetchNameValue** (const char \*pszKey) const
- const char \* **FetchNameValueDef** (const char \*pszKey, const char \*pszDefault) const
- **CPLStringList & AddNameValue** (const char \*pszKey, const char \*pszValue)
- **CPLStringList & SetNameValue** (const char \*pszKey, const char \*pszValue)
- **CPLStringList & Assign** (char \*\*papszList, int bTakeOwnership=1)
- char \* **operator[]** (int i)
- char \*\* **StealList** ()
- **CPLStringList & Sort** ()

### 12.27.1 Detailed Description

String list class designed around our use of C "char\*" string lists.

### 12.27.2 Constructor & Destructor Documentation

12.27.2.1 **CPLStringList::CPLStringList** ( char \*\* *papszListIn*, int *bTakeOwnership* = 1 )

**CPLStringList** (p. ??) constructor.

Parameters

<i>papszListIn</i>	the NULL terminated list of strings to consume.
<i>bTakeOwnership</i>	TRUE if the <b>CPLStringList</b> (p. ??) should take ownership of the list of strings which implies responsibility to free them.

References **Assign**().

### 12.27.3 Member Function Documentation

12.27.3.1 **CPLStringList & CPLStringList::AddNameValue** ( const char \* *pszKey*, const char \* *pszValue* )

A a name=value entry to the list.

A key=value string is prepared and appended to the list. There is no check for other values for the same key in the list.



## Parameters

<i>pszKey</i>	the key name to add.
<i>pszValue</i>	the key value to add.

References AddStringDirectly(), CPLMalloc(), and InsertStringDirectly().

Referenced by SetNameValue().

### 12.27.3.2 CPLStringList & CPLStringList::AddString ( const char \* *pszNewString* )

Add a string to the list.

A copy of the passed in string is made and inserted in the list.

## Parameters

<i>pszNewString</i>	the string to add to the list.
---------------------	--------------------------------

References AddStringDirectly(), and CPLStrdup().

Referenced by CSLTokenizeString2(), GOA2GetAccessToken(), GOA2GetRefreshToken(), and VSIReadDirRecursive().

### 12.27.3.3 CPLStringList & CPLStringList::AddStringDirectly ( char \* *pszNewString* )

Add a string to the list.

This method is similar to **AddString()** (p. ??), but ownership of the *pszNewString* is transferred to the **CPLStringList** (p. ??) class.

## Parameters

<i>pszNewString</i>	the string to add to the list.
---------------------	--------------------------------

References Count().

Referenced by AddNameValue(), and AddString().

### 12.27.3.4 CPLStringList & CPLStringList::Assign ( char \*\* *papszListIn*, int *bTakeOwnership* = 1 )

Assign a list of strings.

## Parameters

<i>papszListIn</i>	the NULL terminated list of strings to consume.
<i>bTakeOwnership</i>	TRUE if the <b>CPLStringList</b> (p. ??) should take ownership of the list of strings which implies responsibility to free them.

## Returns

a reference to the **CPLStringList** (p. ??) on which it was invoked.

References Clear().

Referenced by CPLStringList(), and CSLTokenizeString2().

### 12.27.3.5 CPLStringList & CPLStringList::Clear ( )

Clear the string list.

References CSLDestroy().

Referenced by Assign().

### 12.27.3.6 int CPLStringList::Count ( ) const

#### Returns

count of strings in the list, zero if empty.

References CSLCount().

Referenced by AddStringDirectly(), CSLTokenizeString2(), InsertStringDirectly(), operator[](), SetNameValue(), and Sort().

### 12.27.3.7 int CPLStringList::FetchBoolean ( const char \* pszKey, int bDefault ) const

Check for boolean key value.

In a **CPLStringList** (p. ??) of "Name=Value" pairs, look to see if there is a key with the given name, and if it can be interpreted as being TRUE. If the key appears without any "=Value" portion it will be considered true. If the value is NO, FALSE or 0 it will be considered FALSE otherwise if the key appears in the list it will be considered TRUE. If the key doesn't appear at all, the indicated default value will be returned.

#### Parameters

<i>pszKey</i>	the key value to look for (case insensitive).
<i>bDefault</i>	the value to return if the key isn't found at all.

#### Returns

TRUE or FALSE

References CSLTestBoolean(), and FetchNameValue().

### 12.27.3.8 const char \* CPLStringList::FetchNameValue ( const char \* pszName ) const

Fetch value associated with this key name.

If this list sorted, a fast binary search is done, otherwise a linear scan is done. Name lookup is case insensitive.

#### Parameters

<i>pszName</i>	the key name to search for.
----------------	-----------------------------

#### Returns

the corresponding value or NULL if not found. The returned string should not be modified and points into internal object state that may change on future calls.

References FindName().

Referenced by FetchBoolean(), and FetchNameValueDef().

### 12.27.3.9 const char \* CPLStringList::FetchNameValueDef ( const char \* pszName, const char \* pszDefault ) const

Fetch value associated with this key name.

If this list sorted, a fast binary search is done, otherwise a linear scan is done. Name lookup is case insensitive.

## Parameters

<i>pszName</i>	the key name to search for.
<i>pszDefault</i>	the default value returned if the named entry isn't found.

## Returns

the corresponding value or the passed default if not found.

References `FetchNameValue()`.

Referenced by `GOA2GetAccessToken()`, and `GOA2GetRefreshToken()`.

12.27.3.10 `int CPLStringList::FindName ( const char * pszKey ) const`

Get index of given name/value keyword.

Note that this search is for a line in the form name=value or name:value. Use `FindString()` or `PartialFindString()` for searches not based on name=value pairs.

## Parameters

<i>pszKey</i>	the name to search for.
---------------	-------------------------

## Returns

the string list index of this name, or -1 on failure.

References `CSLFindName()`.

Referenced by `FetchNameValue()`, and `SetNameValue()`.

12.27.3.11 `CPLStringList * CPLStringList::InsertString ( int nInsertAtLineNo, const char * pszNewLine ) [inline]`

Insert into the list at identified location.

This method will insert a string into the list at the identified location. The insertion point must be within or at the end of the list. The following entries are pushed down to make space.

## Parameters

<i>nInsertAtLineNo</i>	the line to insert at, zero to insert at front.
<i>pszNewLine</i>	to the line to insert. This string will be copied.

References `CPLStrdup()`.

12.27.3.12 `CPLStringList & CPLStringList::InsertStringDirectly ( int nInsertAtLineNo, char * pszNewLine )`

Insert into the list at identified location.

This method will insert a string into the list at the identified location. The insertion point must be within or at the end of the list. The following entries are pushed down to make space.

## Parameters

<i>nInsertAtLineNo</i>	the line to insert at, zero to insert at front.
<i>pszNewLine</i>	to the line to insert, the ownership of this string will be taken over the by the object. It must have been allocated on the heap.

References Count(), and CPLError().

Referenced by AddNameValue().

#### 12.27.3.13 char \* CPLStringList::operator[] ( int *i* )

Fetch entry "*i*".

Fetches the requested item in the list. Note that the returned string remains owned by the **CPLStringList** (p. ??). If "*i*" is out of range NULL is returned.

##### Parameters

<i>i</i>	the index of the list item to return.
----------	---------------------------------------

##### Returns

selected entry in the list.

References Count().

#### 12.27.3.14 CPLStringList & CPLStringList::SetNameValue ( const char \* *pszKey*, const char \* *pszValue* )

Set name=value entry in the list.

Similar to **AddNameValue()** (p. ??), except if there is already a value for the key in the list it is replaced instead of adding a new entry to the list. If *pszValue* is NULL any existing key entry is removed.

##### Parameters

<i>pszKey</i>	the key name to add.
<i>pszValue</i>	the key value to add.

References AddNameValue(), Count(), CPLMalloc(), and FindName().

#### 12.27.3.15 CPLStringList & CPLStringList::Sort ( )

Sort the entries in the list and mark list sorted.

Note that once put into "sorted" mode, the **CPLStringList** (p. ??) will attempt to keep things in sorted order through calls to **AddString()** (p. ??), **AddStringDirectly()** (p. ??), **AddNameValue()** (p. ??), **SetNameValue()** (p. ??). Complete list assignments (via **Assign()** (p. ??) and operator= will clear the sorting state. When in sorted order **FindName()** (p. ??), **FetchNameValue()** (p. ??) and **FetchNameValueDef()** (p. ??) will do a binary search to find the key, substantially improve lookup performance in large lists.

References Count().

#### 12.27.3.16 char \*\* CPLStringList::StealList ( )

Seize ownership of underlying string array.

This method is simimilar to List(), except that the returned list is now owned by the caller and the **CPLStringList** (p. ??) is emptied.

**Returns**

the C style string list.

Referenced by CSLTokenizeString2(), and VSIRReadDirRecursive().

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

- **cpl\_string.h**
- cplstringlist.cpp

## 12.28 CPLXMLNode Struct Reference

```
#include <cpl_minixml.h>
```

**Public Attributes**

- **CPLXMLNodeType eType**  
*Node type.*
- **char \* pszValue**  
*Node value.*
- **struct CPLXMLNode \* psNext**  
*Next sibling.*
- **struct CPLXMLNode \* psChild**  
*Child node.*

### 12.28.1 Detailed Description

Document node structure.

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

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

### 12.28.2 Member Data Documentation

#### 12.28.2.1 CPLXMLNodeType CPLXMLNode::eType

Node type.

One of CXT\_Element, CXT\_Text, CXT\_Attribute, CXT\_Comment, or CXT\_Literal.

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

#### 12.28.2.2 struct CPLXMLNode\* CPLXMLNode::psChild

Child node.

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

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

#### 12.28.2.3 struct CPLXMLNode\* CPLXMLNode::psNext

Next sibling.

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

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

#### 12.28.2.4 char\* CPLXMLNode::pszValue

Node value.

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

For CXT\_Attribute the pszValue is the attribute name. The value of the attribute will be a CXT\_Text child.

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

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

For CXT\_Comment the value is all the literal text within the comment, but not including the comment start/end indicators ("<--" and "-->").

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

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

- `cpl_minixml.h`

## 12.29 CPLZip Struct Reference

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

- `cpl_minizip_zip.cpp`

## 12.30 ctb Struct Reference

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

- `cpl_csv.cpp`

## 12.31 curfile\_info Struct Reference

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

- cpl\_minizip\_zip.cpp

## 12.32 DefaultCSVFileNameTLS Struct Reference

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

- cpl\_csv.cpp

## 12.33 errHandler Struct Reference

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

- cpl\_error.cpp

## 12.34 file\_in\_zip\_read\_info\_s Struct Reference

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

- cpl\_minizip\_unzip.cpp

## 12.35 FindFileTLS Struct Reference

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

- cpl\_findfile.cpp

## 12.36 GDALScaledProgressInfo Struct Reference

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

- cpl\_progress.cpp

## 12.37 GZipSnapshot Struct Reference

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

- cpl\_vsif\_gzip.cpp

## 12.38 linkedlist\_data\_s Struct Reference

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

- cpl\_minizip\_zip.cpp

## 12.39 linkedlist\_datablock\_internal\_s Struct Reference

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

- cpl\_minizip\_zip.cpp

## 12.40 OGR\_SRSNode Class Reference

```
#include <ogr_spatialref.h>
```

### Public Member Functions

- **OGR\_SRSNode** (const char \*=NULL)
- int **GetChildCount** () const
- **OGR\_SRSNode \*** **GetChild** (int)
- **OGR\_SRSNode \*** **GetNode** (const char \*)
- void **InsertChild** (**OGR\_SRSNode** \*, int)
- void **AddChild** (**OGR\_SRSNode** \*)
- int **FindChild** (const char \*) const
- void **DestroyChild** (int)
- void **StripNodes** (const char \*)
- const char \* **GetValue** () const
- void **SetValue** (const char \*)
- void **MakeValueSafe** ()
- OGRErr **FixupOrdering** ()
- **OGR\_SRSNode \*** **Clone** () const
- OGRErr **importFromWkt** (char \*\*)
- OGRErr **exportToWkt** (char \*\*) const
- OGRErr **applyRemapper** (const char \*pszNode, char \*\*papszSrcValues, char \*\*papszDstValues, int nStepSize=1, int bChildOfHit=FALSE)

### 12.40.1 Detailed Description

Objects of this class are used to represent value nodes in the parsed representation of the WKT SRS format. For instance UNIT["METER",1] would be rendered into three OGR\_SRSNodes. The root node would have a value of UNIT, and two children, the first with a value of METER, and the second with a value of 1.

Normally application code just interacts with the **OGRSpatialReference** (p. ??) object, which uses the **OGR\_SRSNode** (p. ??) to implement it's data structure; however, this class is user accessible for detailed access to components of an SRS definition.

### 12.40.2 Constructor & Destructor Documentation

#### 12.40.2.1 OGR\_SRSNode::OGR\_SRSNode ( const char \* pszValueIn = NULL )

Constructor.

Parameters

---



<i>pszValueIn</i>	this optional parameter can be used to initialize the value of the node upon creation. If omitted the node will be created with a value of "". Newly created OGR_SRSNodes have no children.
-------------------	---

References CPLStrdup().

Referenced by Clone().

### 12.40.3 Member Function Documentation

#### 12.40.3.1 void OGR\_SRSNode::AddChild ( OGR\_SRSNode \* poNew )

Add passed node as a child of target node.

Note that ownership of the passed node is assumed by the node on which the method is invoked ... use the **Clone()** (p. ??) method if the original is to be preserved. New children are always added at the end of the list.

Parameters

<i>poNew</i>	the node to add as a child.
--------------	-----------------------------

References InsertChild().

Referenced by Clone(), OGRSpatialReference::CloneGeogCS(), OGRSpatialReference::importFromCRSURL(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::importFromURN(), OGRSpatialReference::importFromWkt(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGRSpatialReference::SetAngularUnits(), OGRSpatialReference::SetAuthority(), OGRSpatialReference::SetAxes(), OGRSpatialReference::SetCompoundCS(), OGRSpatialReference::SetExtension(), OGRSpatialReference::SetFromUserInput(), OGRSpatialReference::SetGeogCS(), OGRSpatialReference::SetNode(), OGRSpatialReference::SetProjParm(), OGRSpatialReference::SetTargetLinearUnits(), OGRSpatialReference::SetTOWGS84(), and OGRSpatialReference::SetVertCS().

#### 12.40.3.2 OGRErr OGR\_SRSNode::applyRemapper ( const char \* pszNode, char \*\* papszSrcValues, char \*\* papszDstValues, int nStepSize = 1, int bChildOfHit = FALSE )

Remap node values matching list.

Remap the value of this node or any of it's children if it matches one of the values in the source list to the corresponding value from the destination list. If the pszNode value is set, only do so if the parent node matches that value. Even if a replacement occurs, searching continues.

Parameters

<i>pszNode</i>	Restrict remapping to children of this type of node (eg. "PROJECTION")
<i>papszSrcValues</i>	a NULL terminated array of source string. If the node value matches one of these (case insensitive) then replacement occurs.
<i>papszDstValues</i>	an array of destination strings. On a match, the one corresponding to a source value will be used to replace a node.
<i>nStepSize</i>	increment when stepping through source and destination arrays, allowing source and destination arrays to be one interleaved array for instances. Defaults to 1.
<i>bChildOfHit</i>	Only TRUE if we the current node is the child of a match, and so needs to be set. Application code would normally pass FALSE for this argument.

Returns

returns OGRERR\_NONE unless something bad happens. There is no indication returned about whether any replacement occurred.

References applyRemapper(), GetChild(), GetChildCount(), and SetValue().

Referenced by applyRemapper(), OGRSpatialReference::morphFromESRI(), and OGRSpatialReference::morphToESRI().

### 12.40.3.3 OGR\_SRSNode \* OGR\_SRSNode::Clone ( ) const

Make a duplicate of this node, and it's children.

#### Returns

a new node tree, which becomes the responsibility of the caller.

References AddChild(), and OGR\_SRSNode().

Referenced by OGRSpatialReference::Clone(), OGRSpatialReference::CloneGeogCS(), OGRSpatialReference::CopyGeogCSFrom(), OGRSpatialReference::importFromCRSURL(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::importFromURN(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::SetCompoundCS(), OGRSpatialReference::SetFromUserInput(), OGRSpatialReference::SetGeogCS(), and OGRSpatialReference::StripVertical().

### 12.40.3.4 void OGR\_SRSNode::DestroyChild ( int *iChild* )

Remove a child node, and it's subtree.

Note that removing a child node will result in children after it being renumbered down one.

#### Parameters

<i>iChild</i>	the index of the child.
---------------	-------------------------

Referenced by OGRSpatialReference::CopyGeogCSFrom(), OGRSpatialReference::importFromESRI(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGRSpatialReference::SetAuthority(), OGRSpatialReference::SetAxes(), OGRSpatialReference::SetGeogCS(), OGRSpatialReference::SetStatePlane(), OGRSpatialReference::SetTargetLinearUnits(), OGRSpatialReference::SetTOWGS84(), and StripNodes().

### 12.40.3.5 OGRErr OGR\_SRSNode::exportToWkt ( char \*\* *ppszResult* ) const

Convert this tree of nodes into WKT format.

Note that the returned WKT string should be freed with OGRFree() or CPLFree() when no longer needed. It is the responsibility of the caller.

#### Parameters

<i>ppszResult</i>	the resulting string is returned in this pointer.
-------------------	---

#### Returns

currently OGRErr\_NONE is always returned, but the future it is possible error conditions will develop.

References CPLCalloc(), CPLMalloc(), CSLDestroy(), and exportToWkt().

Referenced by exportToWkt(), and OGRSpatialReference::exportToWkt().

### 12.40.3.6 int OGR\_SRSNode::FindChild ( const char \* *pszValue* ) const

Find the index of the child matching the given string.

Note that the node value must match pszValue with the exception of case. The comparison is case insensitive.

## Parameters

<i>pszValue</i>	the node value being searched for.
-----------------	------------------------------------

## Returns

the child index, or -1 on failure.

Referenced by OGRSpatialReference::CopyGeogCSFrom(), OGRSpatialReference::Fixup(), OGRSpatialReference::GetAuthorityCode(), OGRSpatialReference::GetAuthorityName(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::SetAngularUnits(), OGRSpatialReference::SetAuthority(), OGRSpatialReference::SetAxes(), OGRSpatialReference::SetGeogCS(), OGRSpatialReference::SetStatePlane(), OGRSpatialReference::SetTargetLinearUnits(), OGRSpatialReference::SetTOWGS84(), and StripNodes().

## 12.40.3.7 OGRErr OGR\_SRSNode::FixupOrdering ( )

Correct parameter ordering to match CT Specification.

Some mechanisms to create WKT using **OGRSpatialReference** (p. ??), and some imported WKT fail to maintain the order of parameters required according to the BNF definitions in the OpenGIS SF-SQL and CT Specifications. This method attempts to massage things back into the required order.

This method will reorder the children of the node it is invoked on and then recurse to all children to fix up their children.

## Returns

OGRErr\_NONE on success or an error code if something goes wrong.

References CPLCalloc(), CPLDebug(), CSLFindString(), FixupOrdering(), GetChild(), GetChildCount(), and GetValue().

Referenced by FixupOrdering(), and OGRSpatialReference::FixupOrdering().

## 12.40.3.8 OGR\_SRSNode \* OGR\_SRSNode::GetChild ( int iChild )

Fetch requested child.

## Parameters

<i>iChild</i>	the index of the child to fetch, from 0 to <b>GetChildCount()</b> (p. ??) - 1.
---------------	--

## Returns

a pointer to the child **OGR\_SRSNode** (p. ??), or NULL if there is no such child.

Referenced by applyRemapper(), OGRSpatialReference::EPSGTreatsAsLatLong(), OGRSpatialReference::EPSGTreatsAsNorthingEasting(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToProj4(), OGRSpatialReference::FindProjParm(), FixupOrdering(), OGRSpatialReference::GetAngularUnits(), OGRSpatialReference::GetAttrValue(), OGRSpatialReference::GetAuthorityCode(), OGRSpatialReference::GetAuthorityName(), OGRSpatialReference::GetAxis(), OGRSpatialReference::GetExtension(), OGRSpatialReference::GetInvFlattening(), OGRSpatialReference::GetPrimeMeridian(), OGRSpatialReference::GetProjParm(), OGRSpatialReference::GetSemiMajor(), OGRSpatialReference::GetTargetLinearUnits(), OGRSpatialReference::GetTOWGS84(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::importFromURN(), OGRSpatialReference::IsSame(), MakeValueSafe(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGRSpatialReference::SetAngularUnits(), OGRSpatialReference::SetExtension(), OGRSpatialReference::SetFromUserInput(), OGRSpatialReference::SetLinearUnitsAndUpdateParameters(), OGRSpatialReference::SetNode(), OGRSpatialReference::SetProjParm(), OGRSpatialReference::SetTargetLinearUnits(), StripNodes(), and OGRSpatialReference::StripVertical().

### 12.40.3.9 int OGR\_SRSNode::GetChildCount ( ) const [inline]

Get number of children nodes.

#### Returns

0 for leaf nodes, or the number of children nodes.

Referenced by `applyRemapper()`, `OGRSpatialReference::EPSGTreatsAsLatLong()`, `OGRSpatialReference::EPSGTreatsAsNorthingEasting()`, `OGRSpatialReference::exportToPCI()`, `OGRSpatialReference::exportToProj4()`, `OGRSpatialReference::FindProjParm()`, `FixupOrdering()`, `OGRSpatialReference::GetAngularUnits()`, `OGRSpatialReference::GetAttrValue()`, `OGRSpatialReference::GetAuthorityCode()`, `OGRSpatialReference::GetAuthorityName()`, `OGRSpatialReference::GetAxis()`, `OGRSpatialReference::GetExtension()`, `OGRSpatialReference::GetInvFlattening()`, `OGRSpatialReference::GetPrimeMeridian()`, `OGRSpatialReference::GetSemiMajor()`, `OGRSpatialReference::GetTargetLinearUnits()`, `OGRSpatialReference::GetTOWGS84()`, `OGRSpatialReference::importFromProj4()`, `OGRSpatialReference::IsSame()`, `MakeValueSafe()`, `OGRSpatialReference::morphToESRI()`, `OGRSpatialReference::SetAngularUnits()`, `OGRSpatialReference::SetExtension()`, `OGRSpatialReference::SetLinearUnitsAndUpdateParameters()`, `OGRSpatialReference::SetNode()`, `OGRSpatialReference::SetProjParm()`, `OGRSpatialReference::SetTargetLinearUnits()`, `OGRSpatialReference::SetTOWGS84()`, and `StripNodes()`.

### 12.40.3.10 OGR\_SRSNode \* OGR\_SRSNode::GetNode ( const char \* pszName )

Find named node in tree.

This method does a pre-order traversal of the node tree searching for a node with this exact value (case insensitive), and returns it. Leaf nodes are not considered, under the assumption that they are just attribute value nodes.

If a node appears more than once in the tree (such as UNIT for instance), the first encountered will be returned. Use `GetNode()` (p. ??) on a subtree to be more specific.

#### Parameters

<i>pszName</i>	the name of the node to search for.
----------------	-------------------------------------

#### Returns

a pointer to the node found, or NULL if none.

References `GetNode()`.

Referenced by `OGRSpatialReference::exportToProj4()`, `OGRSpatialReference::GetAttrNode()`, `GetNode()`, `OGRSpatialReference::importFromProj4()`, and `OGRSpatialReference::SetGeocCS()`.

### 12.40.3.11 const char \* OGR\_SRSNode::GetValue ( ) const [inline]

Fetch value string for this node.

#### Returns

A non-NULL string is always returned. The returned pointer is to the internal value of this node, and should not be modified, or freed.

Referenced by `OGRSpatialReference::EPSGTreatsAsLatLong()`, `OGRSpatialReference::EPSGTreatsAsNorthingEasting()`, `OGRSpatialReference::exportToPCI()`, `OGRSpatialReference::exportToProj4()`, `OGRSpatialReference::FindProjParm()`, `FixupOrdering()`, `OGRSpatialReference::GetAngularUnits()`, `OGRSpatialReference::GetAttrValue()`, `OGRSpatialReference::GetAuthorityCode()`, `OGRSpatialReference::GetAuthorityName()`, `OGRSpatialReference::GetAxis()`, `OGRSpatialReference::GetExtension()`, `OGRSpatialReference::GetInvFlattening()`, `OGRSpatialReference::GetPrimeMeridian()`, `OGRSpatialReference::GetProjParm()`, `OGRSpatialReference::GetSemiMajor()`, `OGRSpatialReference::GetTargetLinearUnits()`, `OGRSpatialReference::GetTOWGS84()`, `OGRSpatialReference::importFromCRSURL()`, `OGRSpatialReference::importFromProj4()`, `OGRSpatialReference::importFrom-`

URN(), OGRSpatialReference::IsCompound(), OGRSpatialReference::IsGeocentric(), OGRSpatialReference::IsGeographic(), OGRSpatialReference::IsProjected(), OGRSpatialReference::IsSame(), OGRSpatialReference::IsVertical(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGRSpatialReference::SetExtension(), OGRSpatialReference::SetFromUserInput(), OGRSpatialReference::SetGeocCS(), OGRSpatialReference::SetLinearUnitsAndUpdateParameters(), OGRSpatialReference::SetNode(), OGRSpatialReference::SetProjCS(), OGRSpatialReference::SetProjection(), OGRSpatialReference::SetProjParm(), OGRSpatialReference::SetVertCS(), and OGRSpatialReference::StripCTParms().

#### 12.40.3.12 OGRErr OGR\_SRSNode::importFromWkt ( char \*\* *ppszInput* )

Import from WKT string.

This method will wipe the existing children and value of this node, and reassign them based on the contents of the passed WKT string. Only as much of the input string as needed to construct this node, and it's children is consumed from the input string, and the input string pointer is then updated to point to the remaining (unused) input.

##### Parameters

<i>ppszInput</i>	Pointer to pointer to input. The pointer is updated to point to remaining unused input text.
------------------	--

##### Returns

OGRERR\_NONE if import succeeds, or OGRERR\_CORRUPT\_DATA if it fails for any reason.

#### 12.40.3.13 void OGR\_SRSNode::InsertChild ( OGR\_SRSNode \* *poNew*, int *iChild* )

Insert the passed node as a child of target node, at the indicated position.

Note that ownership of the passed node is assumed by the node on which the method is invoked ... use the **Clone()** (p. ??) method if the original is to be preserved. All existing children at location *iChild* and beyond are push down one space to make space for the new child.

##### Parameters

<i>poNew</i>	the node to add as a child.
<i>iChild</i>	position to insert, use 0 to insert at the beginning.

References CPLRealloc().

Referenced by AddChild(), OGRSpatialReference::CopyGeogCSFrom(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::SetGeocCS(), OGRSpatialReference::SetGeogCS(), OGRSpatialReference::SetProjCS(), OGRSpatialReference::SetProjection(), and OGRSpatialReference::SetTOWGS84().

#### 12.40.3.14 void OGR\_SRSNode::MakeValueSafe ( )

Massage value string, stripping special characters so it will be a database safe string.

The operation is also applies to all subnodes of the current node.

References GetChild(), GetChildCount(), and MakeValueSafe().

Referenced by MakeValueSafe().

#### 12.40.3.15 void OGR\_SRSNode::SetValue ( const char \* *pszNewValue* )

Set the node value.

## Parameters

<i>pszNewValue</i>	the new value to assign to this node. The passed string is duplicated and remains the responsibility of the caller.
--------------------	---

References CPLStrdup().

Referenced by applyRemapper(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGRSpatialReference::SetAngularUnits(), OGRSpatialReference::SetExtension(), OGRSpatialReference::SetNode(), OGRSpatialReference::SetProjParm(), and OGRSpatialReference::SetTargetLinearUnits().

#### 12.40.3.16 void OGR\_SRSNode::StripNodes ( const char \* *pszName* )

Strip child nodes matching name.

Removes any decendent nodes of this node that match the given name. Of course children of removed nodes are also discarded.

## Parameters

<i>pszName</i>	the name for nodes that should be removed.
----------------	--

References DestroyChild(), FindChild(), GetChild(), GetChildCount(), and StripNodes().

Referenced by OGRSpatialReference::exportToPrettyWkt(), OGRSpatialReference::importFromEPSG(), OGRSpatialReference::StripCTParms(), and StripNodes().

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

- **ogr\_spatialref.h**
- **ogr\_srsnode.cpp**

## 12.41 ogr\_style\_param Struct Reference

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

- **ogr\_featurestyle.h**

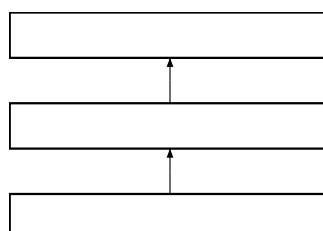
## 12.42 ogr\_style\_value Struct Reference

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

- **ogr\_featurestyle.h**

## 12.43 OGRAbstractProxiedLayer Class Reference

Inheritance diagram for OGRAbstractProxiedLayer:



## Friends

- class **OGRLayerPool**

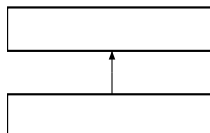
## Additional Inherited Members

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

- ogrlayerpool.h
- ogrlayerpool.cpp

## 12.44 OGRAttrIndex Class Reference

Inheritance diagram for OGRAttrIndex:



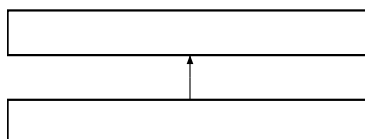
The documentation for this class was generated from the following files:

- ogr\_attrind.h
- ogr\_attrind.cpp

## 12.45 OGRCoordinateTransformation Class Reference

```
#include <ogr_spatialref.h>
```

Inheritance diagram for OGRCoordinateTransformation:



## Public Member Functions

- virtual **OGRSpatialReference** \* **GetSourceCS** ()=0
- virtual **OGRSpatialReference** \* **GetTargetCS** ()=0
- virtual int **Transform** (int nCount, double \*x, double \*y, double \*z=NULL)=0
- virtual int **TransformEx** (int nCount, double \*x, double \*y, double \*z=NULL, int \*pabSuccess=NULL)=0

## Static Public Member Functions

- static void **DestroyCT** (**OGRCoordinateTransformation** \*poCT)  
**OGRCoordinateTransformation** (p. ??) destructor.

### 12.45.1 Detailed Description

Interface for transforming between coordinate systems.

Currently, the only implementation within OGR is **OGRProj4CT** (p. ??), which requires the PROJ.4 library to be available at run-time.

Also, see **OGRCreateCoordinateTransformation()** (p. ??) for creating transformations.

### 12.45.2 Member Function Documentation

12.45.2.1 `void OGRCoordinateTransformation::DestroyCT ( OGRCoordinateTransformation * poCT ) [static]`

**OGRCoordinateTransformation** (p. ??) destructor.

This function is the same as `OGRCoordinateTransformation::~~OGRCoordinateTransformation()` and **OCTDestroyCoordinateTransformation()** (p. ??)

This static method will destroy a **OGRCoordinateTransformation** (p. ??). It is equivalent to calling delete on the object, but it ensures that the deallocation is properly executed within the OGR libraries heap on platforms where this can matter (win32).

#### Parameters

<i>poCT</i>	the object to delete
-------------	----------------------

#### Since

GDAL 1.7.0

12.45.2.2 `virtual OGRSpatialReference* OGRCoordinateTransformation::GetSourceCS ( ) [pure virtual]`

Fetch internal source coordinate system.

Implemented in **OGRProj4CT** (p. ??).

12.45.2.3 `virtual OGRSpatialReference* OGRCoordinateTransformation::GetTargetCS ( ) [pure virtual]`

Fetch internal target coordinate system.

Implemented in **OGRProj4CT** (p. ??).

Referenced by `OGRPoint::transform()`, `OGRLineString::transform()`, `OGRPolygon::transform()`, and `OGRGeometryCollection::transform()`.

12.45.2.4 `virtual int OGRCoordinateTransformation::Transform ( int nCount, double * x, double * y, double * z = NULL ) [pure virtual]`

Transform points from source to destination space.

This method is the same as the C function `OCTTransform()`.

The method **TransformEx()** (p. ??) allows extended success information to be captured indicating which points failed to transform.

#### Parameters



<i>nCount</i>	number of points to transform.
<i>x</i>	array of nCount X vertices, modified in place.
<i>y</i>	array of nCount Y vertices, modified in place.
<i>z</i>	array of nCount Z vertices, modified in place.

**Returns**

TRUE on success, or FALSE if some or all points fail to transform.

Implemented in **OGRProj4CT** (p. ??).

Referenced by OGRPoint::transform().

**12.45.2.5** `virtual int OGRCoordinateTransformation::TransformEx ( int nCount, double * x, double * y, double * z = NULL, int * pabSuccess = NULL ) [pure virtual]`

Transform points from source to destination space.

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

**Parameters**

<i>nCount</i>	number of points to transform.
<i>x</i>	array of nCount X vertices, modified in place.
<i>y</i>	array of nCount Y vertices, modified in place.
<i>z</i>	array of nCount Z vertices, modified in place.
<i>pabSuccess</i>	array of per-point flags set to TRUE if that point transforms, or FALSE if it does not.

**Returns**

TRUE if some or all points transform successfully, or FALSE if if none transform.

Implemented in **OGRProj4CT** (p. ??).

Referenced by OGRLineString::transform().

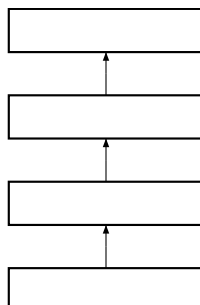
The documentation for this class was generated from the following files:

- **ogr\_spatialref.h**
- ogrct.cpp

## 12.46 OGRCurve Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRCurve:



## Public Member Functions

- virtual double **get\_Length** () const =0  
*Returns the length of the curve.*
- virtual void **StartPoint** (**OGRPoint** \*) const =0  
*Return the curve start point.*
- virtual void **EndPoint** (**OGRPoint** \*) const =0  
*Return the curve end point.*
- virtual int **get\_IsClosed** () const  
*Return TRUE if curve is closed.*
- virtual void **Value** (double, **OGRPoint** \*) const =0  
*Fetch point at given distance along curve.*

### 12.46.1 Detailed Description

Abstract curve base class.

### 12.46.2 Member Function Documentation

#### 12.46.2.1 void OGRCurve::EndPoint ( OGRPoint \* *poPoint* ) const [pure virtual]

Return the curve end point.

This method relates to the SF COM ICurve::get\_EndPoint() method.

##### Parameters

<i>poPoint</i>	the point to be assigned the end location.
----------------	--

Implemented in **OGRLineString** (p. ??).

Referenced by get\_IsClosed().

#### 12.46.2.2 int OGRCurve::get\_IsClosed ( ) const [virtual]

Return TRUE if curve is closed.

Tests if a curve is closed. A curve is closed if its start point is equal to its end point.

This method relates to the SFCOM ICurve::get\_IsClosed() method.

##### Returns

TRUE if closed, else FALSE.

References EndPoint(), OGRPoint::getX(), OGRPoint::getY(), and StartPoint().

#### 12.46.2.3 double OGRCurve::get\_Length ( ) const [pure virtual]

Returns the length of the curve.

This method relates to the SFCOM ICurve::get\_Length() method.

##### Returns

the length of the curve, zero if the curve hasn't been initialized.

Implemented in **OGRLineString** (p. ??).

12.46.2.4 void OGRCurve::StartPoint ( OGRPoint \* *poPoint* ) const [pure virtual]

Return the curve start point.

This method relates to the SF COM ICurve::get\_StartPoint() method.

#### Parameters

<i>poPoint</i>	the point to be assigned the start location.
----------------	--

Implemented in **OGRLineString** (p. ??).

Referenced by get\_IsClosed().

12.46.2.5 void OGRCurve::Value ( double *dfDistance*, OGRPoint \* *poPoint* ) const [pure virtual]

Fetch point at given distance along curve.

This method relates to the SF COM ICurve::get\_Value() method.

#### Parameters

<i>dfDistance</i>	distance along the curve at which to sample position. This distance should be between zero and <b>get_Length()</b> (p. ??) for this curve.
<i>poPoint</i>	the point to be assigned the curve position.

Implemented in **OGRLineString** (p. ??).

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

- **ogr\_geometry.h**
- **ogrcurve.cpp**

## 12.47 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, **OGRwkbGeometryType** 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 \*\*papszOptions=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 \*poResultSet)**  
*Release results of **ExecuteSQL()** (p. ??).*
- 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.*

## Friends

- class **OGRSFDriverRegistrar**

## 12.47.1 Detailed Description

This class represents a data source. A data source potentially consists of many layers (**OGRLayer** (p. ??)). 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** (p. ??) is destroyed, all it's associated OGRLayers objects are also destroyed.

## 12.47.2 Member Function Documentation

**12.47.2.1 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()** (p. ??).

### Parameters

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

### Returns

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

References CPLCalloc(), CPLError(), CPLErrorReset(), CPLMalloc(), OGRLayer::CreateFeature(), OGRFeature::CreateFeature(), OGRLayer::CreateField(), CreateLayer(), OGRFeature::DestroyFeature(), OGRFeature::GetFID(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), OGRFeatureDefn::GetFieldIndex(), OGRFeatureDefn::GetGeomType(), OGRLayer::GetLayerDefn(), OGRFeatureDefn::GetName(), OGRFieldDefn::GetNameRef(), OGRLayer::GetNextFeature(), OGRLayer::GetSpatialRef(), OGRLayer::ResetReading(), OGRFeature::SetFID(), OGRFeature::SetFrom(), OGRLayer::TestCapability(), and TestCapability().

Referenced by OGRSFDriver::CopyDataSource().

**12.47.2.2 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

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

### Returns

NULL is returned on failure, or a new **OGRLayer** (p. ??) handle on success.

### Example:

```
#include "ogr_sfrmts.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 )
    {
        ...
    }

```

References CPLError().

Referenced by CopyLayer().

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

Delete the indicated layer from the datasource.

If this method is supported the ODSDeleteLayer capability will test TRUE on the **OGRDataSource** (p. ??).

This method is the same as the C function **OGR\_DS\_DeleteLayer()** (p. ??).

##### Parameters

<i>iLayer</i>	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.

References CPLError().

#### 12.47.2.4 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.

#### 12.47.2.5 void OGRDataSource::DestroyDataSource ( OGRDataSource \* *poDS* ) [static]

Closes opened datasource and releases allocated resources.

This static method will close and destroy a datasource. It is equivalent 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()** (p. ??).

##### Parameters

<i>poDS</i>	pointer to allocated datasource object.
-------------	---

**12.47.2.6 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 **OGR-Layer** (p. ??) pointer representing a results set from the query. Note that this **OGRLayer** (p. ??) is in addition to the layers in the data store and must be destroyed with **OGRDataSource::ReleaseResultSet()** (p. ??) before the data source is closed (destroyed).

This method is the same as the C function **OGR\_DS\_ExecuteSQL()** (p. ??).

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

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

#### Returns

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

References [CPLError\(\)](#), [CPLRealloc\(\)](#), [CSLCount\(\)](#), and [CSLDestroy\(\)](#).

**12.47.2.7 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()** (p. ??).

#### Returns

NULL if driver info is not available, or pointer to a driver owned by the [OGRSFDriverManager](#).

Referenced by [OGRSFDriver::CopyDataSource\(\)](#), [OGR\\_Dr\\_CopyDataSource\(\)](#), [OGR\\_Dr\\_CreateDataSource\(\)](#), [OGR\\_Dr\\_Open\(\)](#), and [OGRSFDriverRegistrar::Open\(\)](#).

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

Fetch a layer by index.

The returned layer remains owned by the **OGRDataSource** (p. ??) and should not be deleted by the application.

This method is the same as the C function **OGR\_DS\_GetLayer()** (p. ??).

#### Parameters

<i>iLayer</i>	a layer number between 0 and <b>GetLayerCount()</b> (p. ??)-1.
---------------	--

**Returns**

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

Referenced by OGRSFDriver::CopyDataSource(), GetLayerByName(), GetSummaryRefCount(), and SyncToDisk().

#### 12.47.2.9 **OGRLayer \* OGRDataSource::GetLayerByName ( const char \* pszLayerName ) [virtual]**

Fetch a layer by name.

The returned layer remains owned by the **OGRDataSource** (p. ??) and should not be deleted by the application.

This method is the same as the C function **OGR\_DS\_GetLayerByName()** (p. ??).

**Parameters**

<i>pszLayerName</i>	the layer name of the layer to fetch.
---------------------	---------------------------------------

**Returns**

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

References GetLayer(), GetLayerCount(), and OGRLayer::GetName().

#### 12.47.2.10 **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()** (p. ??).

**Returns**

layer count.

Referenced by OGRSFDriver::CopyDataSource(), GetLayerByName(), GetSummaryRefCount(), and SyncToDisk().

#### 12.47.2.11 **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** (p. ??) 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()** (p. ??).

**Returns**

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

#### 12.47.2.12 **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.



**12.47.2.13 OGRStyleTable \* 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.

**12.47.2.14 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.

References `GetLayer()`, `GetLayerCount()`, and `OGRLayer::GetRefCount()`.

**12.47.2.15 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.

Referenced by `OGRSFDriverRegistrar::Open()`.

**12.47.2.16 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()`** (p. ??).

**Returns**

`OGRERR_NONE` on success or an error code.

References `OGRSFDriverRegistrar::GetRegistrar()`.

**12.47.2.17 void OGRDataSource::ReleaseResultSet ( OGRLayer \* poResultSet ) [virtual]**

Release results of **`ExecuteSQL()`** (p. ??).

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

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

## Parameters

<i>poResultsSet</i>	the result of a previous <b>ExecuteSQL()</b> (p. ??) call.
---------------------	--

12.47.2.18 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

<i>poDriver</i>	pointer to driver instance associated with the data source.
-----------------	---

Referenced by OGRSFDriver::CopyDataSource(), OGR\_Dr\_CopyDataSource(), OGR\_Dr\_CreateDataSource(), and OGR\_Dr\_Open().

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

Set data source style table.

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

This method is the same as the C function OGR\_DS\_SetStyleTable().

## Parameters

<i>poStyleTable</i>	pointer to style table to set
---------------------	-------------------------------

References OGRStyleTable::Clone().

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

Set data source style table.

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

This method is the same as the C function OGR\_DS\_SetStyleTableDirectly().

## Parameters

<i>poStyleTable</i>	pointer to style table to set
---------------------	-------------------------------

12.47.2.21 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()** (p. ??) method on each of the layers. Conceptionally, calling **SyncToDisk()** (p. ??) on a datasource should include any work that might be accomplished by calling **SyncToDisk()** (p. ??) on layers in that data source.

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

This method is the same as the C function **OGR\_DS\_SyncToDisk()** (p. ??).

#### Returns

OGRERR\_NONE if no error occurs (even if nothing is done) or an error code.

References GetLayer(), GetLayerCount(), and OGRLayer::SyncToDisk().

**12.47.2.22** `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.

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

This method is the same as the C function **OGR\_DS\_TestCapability()** (p. ??).

#### Parameters

<i>pszCapability</i>	the capability to test.
----------------------	-------------------------

#### Returns

TRUE if capability available otherwise FALSE.

Referenced by CopyLayer().

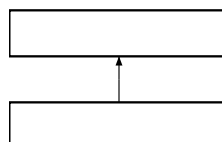
The documentation for this class was generated from the following files:

- **ogrsf\_frmts.h**
- **ogrsf\_frmts.dox**
- **ogrdatasource.cpp**

## 12.48 OGREnvelope Class Reference

```
#include <ogr_core.h>
```

Inheritance diagram for OGREnvelope:



### 12.48.1 Detailed Description

Simple container for a bounding region.

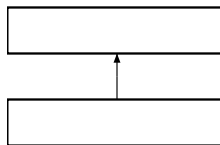
The documentation for this class was generated from the following file:

- **ogr\_core.h**

## 12.49 OGREnvelope3D Class Reference

```
#include <ogr_core.h>
```

Inheritance diagram for OGREnvelope3D:



### 12.49.1 Detailed Description

Simple container for a bounding region in 3D.

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

- **ogr\_core.h**

## 12.50 OGRFeature Class Reference

```
#include <ogr_feature.h>
```

### Public Member Functions

- **OGRFeature (OGRFeatureDefn \*)**  
*Constructor.*
- **OGRFeatureDefn \* GetDefnRef ()**  
*Fetch feature definition.*
- **OGRERR SetGeometryDirectly (OGRGeometry \*)**  
*Set feature geometry.*
- **OGRERR SetGeometry (OGRGeometry \*)**  
*Set feature geometry.*
- **OGRGeometry \* GetGeometryRef ()**  
*Fetch pointer to feature geometry.*
- **OGRGeometry \* StealGeometry ()**  
*Take away ownership of geometry.*
- **OGRFeature \* Clone ()**  
*Duplicate feature.*
- virtual **OGRBoolean Equal (OGRFeature \*poFeature)**  
*Test if two features are the same.*
- **int GetFieldCount ()**

Fetch number of fields on this feature. This will always be the same as the field count for the **OGRFeatureDefn** (p. ??).

- **OGRFieldDefn \* GetFieldDefnRef** (int iField)  
Fetch definition for this field.
- int **GetFieldIndex** (const char \*pszName)  
Fetch the field index given field name.
- int **IsFieldSet** (int iField) const  
Test if a field has ever been assigned a value or not.
- void **UnsetField** (int iField)  
Clear a field, marking it as unset.
- **OGRField \* GetRawFieldRef** (int i)  
Fetch a pointer to the internal field value given the index.
- int **GetFieldAsInteger** (int i)  
Fetch field value as integer.
- double **GetFieldAsDouble** (int i)  
Fetch field value as a double.
- const char \* **GetFieldAsString** (int i)  
Fetch field value as a string.
- const int \* **GetFieldAsIntegerList** (int i, int \*pnCount)  
Fetch field value as a list of integers.
- const double \* **GetFieldAsDoubleList** (int i, int \*pnCount)  
Fetch field value as a list of doubles.
- char \*\* **GetFieldAsStringList** (int i) const  
Fetch field value as a list of strings.
- GByte \* **GetFieldAsBinary** (int i, int \*pnCount)  
Fetch field value as binary data.
- int **GetFieldAsDateTime** (int i, int \*pnYear, int \*pnMonth, int \*pnDay, int \*pnHour, int \*pnMinute, int \*pnSecond, int \*pnTZFlag)  
Fetch field value as date and time.
- void **SetField** (int i, int nValue)  
Set field to integer value.
- void **SetField** (int i, double dfValue)  
Set field to double value.
- void **SetField** (int i, const char \*pszValue)  
Set field to string value.
- void **SetField** (int i, int nCount, int \*panValues)  
Set field to list of integers value.
- void **SetField** (int i, int nCount, double \*padfValues)  
Set field to list of doubles value.
- void **SetField** (int i, char \*\*papszValues)  
Set field to list of strings value.
- void **SetField** (int i, **OGRField** \*puValue)  
Set field.
- void **SetField** (int i, int nCount, GByte \*pabyBinary)  
Set field to binary data.
- void **SetField** (int i, int nYear, int nMonth, int nDay, int nHour=0, int nMinute=0, int nSecond=0, int nTZFlag=0)  
Set field to date.
- long **GetFID** ()  
Get feature identifier.
- virtual OGRErr **SetFID** (long nFID)  
Set the feature identifier.

- void **DumpReadable** (FILE \*, char \*\*papszOptions=NULL)  
*Dump this feature in a human readable form.*
- OGRErr **SetFrom** (OGRFeature \*, int=TRUE)  
*Set one feature from another.*
- OGRErr **SetFrom** (OGRFeature \*, int \*, int=TRUE)  
*Set one feature from another.*
- OGRErr **SetFieldsFrom** (OGRFeature \*, int \*, int=TRUE)  
*Set fields from another feature.*
- virtual const char \* **GetStyleString** ()  
*Fetch style string for this feature.*
- virtual void **SetStyleString** (const char \*)  
*Set feature style string. This method operate exactly as **OGRFeature::SetStyleStringDirectly()** (p. ??) except that it does not assume ownership of the passed string, but instead makes a copy of it.*
- virtual void **SetStyleStringDirectly** (char \*)  
*Set feature style string. This method operate exactly as **OGRFeature::SetStyleString()** (p. ??) except that it assumes ownership of the passed string.*

## Static Public Member Functions

- static OGRFeature \* **CreateFeature** (OGRFeatureDefn \*)  
*Feature factory.*
- static void **DestroyFeature** (OGRFeature \*)  
*Destroy feature.*

### 12.50.1 Detailed Description

A simple feature, including geometry and attributes.

### 12.50.2 Constructor & Destructor Documentation

#### 12.50.2.1 OGRFeature::OGRFeature ( OGRFeatureDefn \* poDefnIn )

Constructor.

Note that the **OGRFeature** (p. ??) will increment the reference count of it's defining **OGRFeatureDefn** (p. ??). Destruction of the **OGRFeatureDefn** (p. ??) before destruction of all OGRFeatures that depend on it is likely to result in a crash.

This method is the same as the C function **OGR\_F\_Create()** (p. ??).

Parameters

<i>poDefnIn</i>	feature class (layer) definition to which the feature will adhere.
-----------------	--

References CPLMalloc(), OGRFeatureDefn::GetFieldCount(), and OGRFeatureDefn::Reference().

Referenced by Clone(), and CreateFeature().

### 12.50.3 Member Function Documentation

#### 12.50.3.1 OGRFeature \* OGRFeature::Clone ( )

Duplicate feature.

The newly created feature is owned by the caller, and will have it's own reference to the **OGRFeatureDefn** (p. ??).

This method is the same as the C function **OGR\_F\_Clone()** (p. ??).

**Returns**

new feature, exactly matching this feature.

References GetFID(), OGRFeatureDefn::GetFieldCount(), GetStyleString(), OGRFeature(), SetFID(), SetField(), SetGeometry(), and SetStyleString().

Referenced by OGRGenSQLResultsLayer::GetFeature().

### 12.50.3.2 OGRFeature \* OGRFeature::CreateFeature ( OGRFeatureDefn \* *poDefn* ) [static]

Feature factory.

This is essentially a feature factory, useful for applications creating features but wanting to ensure they are created out of the OGR/GDAL heap.

This method is the same as the C function **OGR\_F\_Create()** (p. ??).

**Parameters**

<i>poDefn</i>	Feature definition defining schema.
---------------	-------------------------------------

**Returns**

new feature object with null fields and no geometry. May be deleted with delete.

References OGRFeature().

Referenced by OGRDataSource::CopyLayer().

### 12.50.3.3 void OGRFeature::DestroyFeature ( OGRFeature \* *poFeature* ) [static]

Destroy feature.

The feature is deleted, but within the context of the GDAL/OGR heap. This is necessary when higher level applications use GDAL/OGR from a DLL and they want to delete a feature created within the DLL. If the delete is done in the calling application the memory will be freed onto the application heap which is inappropriate.

This method is the same as the C function **OGR\_F\_Destroy()** (p. ??).

**Parameters**

<i>poFeature</i>	the feature to delete.
------------------	------------------------

Referenced by OGRDataSource::CopyLayer().

### 12.50.3.4 void OGRFeature::DumpReadable ( FILE \* *fpOut*, char \*\* *papszOptions* = NULL )

Dump this feature in a human readable form.

This dumps the attributes, and geometry; however, it doesn't definition information (other than field types and names), nor does it report the geometry spatial reference system.

A few options can be defined to change the default dump :

- DISPLAY\_FIELDS=NO : to hide the dump of the attributes
- DISPLAY\_STYLE=NO : to hide the dump of the style string
- DISPLAY\_GEOMETRY=NO : to hide the dump of the geometry
- DISPLAY\_GEOMETRY=SUMMARY : to get only a summary of the geometry

This method is the same as the C function **OGR\_F\_DumpReadable()** (p. ??).

## Parameters

<i>fpOut</i>	the stream to write to, such as stdout. If NULL stdout will be used.
<i>papszOptions</i>	NULL terminated list of options (may be NULL)

References CSLTestBoolean(), OGRGeometry::dumpReadable(), GetFID(), GetFieldAsString(), GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetFieldType(), OGRFeatureDefn::GetName(), OGRFieldDefn::GetNameRef(), GetStyleString(), OGRFieldDefn::GetType(), and IsFieldSet().

### 12.50.3.5 OGRBoolean OGRFeature::Equal ( OGRFeature \* poFeature ) [virtual]

Test if two features are the same.

Two features are considered equal if they share the same (pointer equality) same **OGRFeatureDefn** (p. ??), have the same field values, and the same geometry (as tested by OGRGeometry::Equal()) as well as the same feature id.

This method is the same as the C function **OGR\_F\_Equal()** (p. ??).

## Parameters

<i>poFeature</i>	the other feature to test this one against.
------------------	---

## Returns

TRUE if they are equal, otherwise FALSE.

References CSLCount(), OGRGeometry::Equals(), GetDefnRef(), GetFID(), GetFieldAsBinary(), GetFieldAsDateTime(), GetFieldAsDouble(), GetFieldAsDoubleList(), GetFieldAsInteger(), GetFieldAsIntegerList(), GetFieldAsString(), GetFieldAsStringList(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), GetGeometryRef(), OGRFieldDefn::GetType(), IsFieldSet(), OFTBinary, OFTDate, OFTDateTime, OFTInteger, OFTIntegerList, OFTReal, OFTRealList, OFTString, OFTStringList, and OFTTime.

### 12.50.3.6 OGRFeatureDefn \* OGRFeature::GetDefnRef ( ) [inline]

Fetch feature definition.

This method is the same as the C function **OGR\_F\_GetDefnRef()** (p. ??).

## Returns

a reference to the feature definition object.

Referenced by Equal().

### 12.50.3.7 long OGRFeature::GetFID ( ) [inline]

Get feature identifier.

This method is the same as the C function **OGR\_F\_GetFID()** (p. ??).

## Returns

feature id or OGRNullFID if none has been assigned.

Referenced by Clone(), OGRDataSource::CopyLayer(), OGRUnionLayer::CreateFeature(), DumpReadable(), Equal(), OGRLayer::GetFeature(), GetFieldAsDouble(), GetFieldAsInteger(), GetFieldAsString(), and OGRUnionLayer::SetFeature().



### 12.50.3.8 GByte \* OGRFeature::GetFieldAsBinary ( int *iField*, int \* *pnBytes* )

Fetch field value as binary data.

Currently this method only works for OFTBinary fields.

This method is the same as the C function **OGR\_F\_GetFieldAsBinary()** (p. ??).

#### Parameters

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>pnBytes</i>	location to put the number of bytes returned.

#### Returns

the field value. This data is internal, and should not be modified, or freed. Its lifetime may be very brief.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), and OFTBinary.

Referenced by Equal().

### 12.50.3.9 int OGRFeature::GetFieldAsDateTime ( int *iField*, int \* *pnYear*, int \* *pnMonth*, int \* *pnDay*, int \* *pnHour*, int \* *pnMinute*, int \* *pnSecond*, int \* *pnTZFlag* )

Fetch field value as date and time.

Currently this method only works for OFTDate, OFTTime and OFTDateTime fields.

This method is the same as the C function **OGR\_F\_GetFieldAsDateTime()** (p. ??).

#### Parameters

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>pnYear</i>	(including century)
<i>pnMonth</i>	(1-12)
<i>pnDay</i>	(1-31)
<i>pnHour</i>	(0-23)
<i>pnMinute</i>	(0-59)
<i>pnSecond</i>	(0-59)
<i>pnTZFlag</i>	(0=unknown, 1=localtime, 100=GMT, see data model for details)

#### Returns

TRUE on success or FALSE on failure.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OFTDate, OFTDateTime, and OFTTime.

Referenced by Equal().

### 12.50.3.10 double OGRFeature::GetFieldAsDouble ( int *iField* )

Fetch field value as a double.

OFTString features will be translated using atof(). OFTInteger fields will be cast to double. Other field types, or errors will result in a return value of zero.

This method is the same as the C function **OGR\_F\_GetFieldAsDouble()** (p. ??).

**Parameters**

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
---------------	---

**Returns**

the field value.

References GetFID(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OFTInteger, OFTReal, OFTString, and OGR\_G\_Area().

Referenced by Equal(), and SetFieldsFrom().

### 12.50.3.11 `const double * OGRFeature::GetFieldAsDoubleList ( int iField, int * pnCount )`

Fetch field value as a list of doubles.

Currently this method only works for OFTRealList fields.

This method is the same as the C function **OGR\_F\_GetFieldAsDoubleList()** (p. ??).

**Parameters**

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>pnCount</i>	an integer to put the list count (number of doubles) into.

**Returns**

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief. If \*pnCount is zero on return the returned pointer may be NULL or non-NULL.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), and OFTRealList.

Referenced by Equal(), and SetFieldsFrom().

### 12.50.3.12 `int OGRFeature::GetFieldAsInteger ( int iField )`

Fetch field value as integer.

OFTString features will be translated using atoi(). OFTReal fields will be cast to integer. Other field types, or errors will result in a return value of zero.

This method is the same as the C function **OGR\_F\_GetFieldAsInteger()** (p. ??).

**Parameters**

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
---------------	---

**Returns**

the field value.

References GetFID(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OFTInteger, OFTReal, OFTString, and OGR\_G\_Area().

Referenced by Equal(), and SetFieldsFrom().

### 12.50.3.13 `const int * OGRFeature::GetFieldAsIntegerList ( int iField, int * pnCount )`

Fetch field value as a list of integers.

Currently this method only works for OFTIntegerList fields.

This method is the same as the C function **OGR\_F\_GetFieldAsIntegerList()** (p. ??).

## Parameters

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>pnCount</i>	an integer to put the list count (number of integers) into.

## Returns

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief. If \*pnCount is zero on return the returned pointer may be NULL or non-NULL.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), and OFTIntegerList.

Referenced by Equal(), and SetFieldsFrom().

12.50.3.14 `const char * OGRFeature::GetFieldAsString ( int iField )`

Fetch field value as a string.

OFTReal and OFTInteger fields will be translated to string using sprintf(), but not necessarily using the established formatting rules. Other field types, or errors will result in a return value of zero.

This method is the same as the C function **OGR\_F\_GetFieldAsString()** (p. ??).

## Parameters

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
---------------	---

## Returns

the field value. This string is internal, and should not be modified, or freed. Its lifetime may be very brief.

References CPLBinaryToHex(), CPLStrdup(), OGRGeometry::exportToWkt(), GetFID(), OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), OGRGeometry::getGeometryName(), OGRFieldDefn::GetPrecision(), GetStyleString(), OGRFieldDefn::GetType(), OGRFieldDefn::GetWidth(), IsFieldSet(), OFTBinary, OFTDate, OFTDateTime, OFTInteger, OFTIntegerList, OFTReal, OFTRealList, OFTString, OFTStringList, OFTTime, and OGR\_G\_Area().

Referenced by OGRUnionLayer::CreateFeature(), DumpReadable(), Equal(), GetStyleString(), OGRUnionLayer::SetFeature(), and SetFieldsFrom().

12.50.3.15 `char ** OGRFeature::GetFieldAsStringList ( int iField ) const`

Fetch field value as a list of strings.

Currently this method only works for OFTStringList fields.

The returned list is terminated by a NULL pointer. The number of elements can also be calculated using **CSLCount()** (p. ??).

This method is the same as the C function **OGR\_F\_GetFieldAsStringList()** (p. ??).

## Parameters

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
---------------	---

## Returns

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), and OFTStringList.

Referenced by Equal().

### 12.50.3.16 `int OGRFeature::GetFieldCount ( ) [inline]`

Fetch number of fields on this feature. This will always be the same as the field count for the **OGRFeatureDefn** (p. ??).

This method is the same as the C function **OGR\_F\_GetFieldCount()** (p. ??).

#### Returns

count of fields.

Referenced by `DumpReadable()`, `OGR_F_IsFieldSet()`, `SetFieldsFrom()`, and `SetFrom()`.

### 12.50.3.17 `OGRFieldDefn * OGRFeature::GetFieldDefnRef ( int iField ) [inline]`

Fetch definition for this field.

This method is the same as the C function **OGR\_F\_GetFieldDefnRef()** (p. ??).

#### Parameters

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
---------------	---

#### Returns

the field definition (from the **OGRFeatureDefn** (p. ??)). This is an internal reference, and should not be deleted or modified.

Referenced by `SetFieldsFrom()`, and `SetFrom()`.

### 12.50.3.18 `int OGRFeature::GetFieldIndex ( const char * pszName ) [inline]`

Fetch the field index given field name.

This is a cover for the **OGRFeatureDefn::GetFieldIndex()** (p. ??) method.

This method is the same as the C function **OGR\_F\_GetFieldIndex()** (p. ??).

#### Parameters

<i>pszName</i>	the name of the field to search for.
----------------	--------------------------------------

#### Returns

the field index, or -1 if no matching field is found.

Referenced by `GetStyleString()`, and `SetFrom()`.

### 12.50.3.19 `OGRGeometry * OGRFeature::GetGeometryRef ( ) [inline]`

Fetch pointer to feature geometry.

This method is the same as the C function **OGR\_F\_GetGeometryRef()** (p. ??).

#### Returns

pointer to internal feature geometry. This object should not be modified.

Referenced by `OGRLayer::Clip()`, `OGRWarpedLayer::CreateFeature()`, `Equal()`, `OGRLayer::Erase()`, `OGRLayer::GetExtent()`, `OGRWarpedLayer::GetFeature()`, `OGRWarpedLayer::GetNextFeature()`, `OGRUnionLayer::GetNextFeature()`, `OGRWarpedLayer::SetFeature()`, and `SetFrom()`.

**12.50.3.20 OGRField \* OGRFeature::GetRawFieldRef ( int *iField* ) [inline]**

Fetch a pointer to the internal field value given the index.

This method is the same as the C function **OGR\_F\_GetRawFieldRef()** (p. ??).

**Parameters**

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
---------------	---

**Returns**

the returned pointer is to an internal data structure, and should not be freed, or modified.

Referenced by SetFieldsFrom().

**12.50.3.21 const char \* OGRFeature::GetStyleString ( ) [virtual]**

Fetch style string for this feature.

Set the OGR Feature Style Specification for details on the format of this string, and **ogr\_featurestyle.h** (p. ??) for services available to parse it.

This method is the same as the C function **OGR\_F\_GetStyleString()** (p. ??).

**Returns**

a reference to a representation in string format, or NULL if there isn't one.

References GetFieldAsString(), and GetFieldIndex().

Referenced by Clone(), DumpReadable(), GetFieldAsString(), OGRStyleMgr::InitFromFeature(), and SetFrom().

**12.50.3.22 int OGRFeature::IsFieldSet ( int *iField* ) const**

Test if a field has ever been assigned a value or not.

This method is the same as the C function **OGR\_F\_IsFieldSet()** (p. ??).

**Parameters**

<i>iField</i>	the field to test.
---------------	--------------------

**Returns**

TRUE if the field has been set, otherwise false.

References OGRFeatureDefn::GetFieldCount(), and OGR\_G\_Area().

Referenced by OGRUnionLayer::CreateFeature(), DumpReadable(), Equal(), GetFieldAsBinary(), GetFieldAsDate-Time(), GetFieldAsDouble(), GetFieldAsDoubleList(), GetFieldAsInteger(), GetFieldAsIntegerList(), GetFieldAsString(), GetFieldAsStringList(), OGR\_F\_IsFieldSet(), OGRUnionLayer::SetFeature(), SetField(), SetFieldsFrom(), and UnsetField().

**12.50.3.23 OGRErr OGRFeature::SetFID ( long *nFID* ) [virtual]**

Set the feature identifier.

For specific types of features this operation may fail on illegal features ids. Generally it always succeeds. Feature ids should be greater than or equal to zero, with the exception of OGRNullFID (-1) indicating that the feature id is unknown.

This method is the same as the C function **OGR\_F\_SetFID()** (p. ??).

## Parameters

<i>nFID</i>	the new feature identifier value to assign.
-------------	---

## Returns

On success OGRERR\_NONE, or on failure some other value.

Referenced by Clone(), OGRDataSource::CopyLayer(), OGRUnionLayer::CreateFeature(), OGRGenSQLResultsLayer::GetFeature(), OGRUnionLayer::SetFeature(), and SetFrom().

#### 12.50.3.24 void OGRFeature::SetField ( int *iField*, int *nValue* )

Set field to integer value.

OFTInteger and OFTReal fields will be set directly. OFTString fields will be assigned a string representation of the value, but not necessarily taking into account formatting constraints on this field. Other field types may be unaffected.

This method is the same as the C function **OGR\_F\_SetFieldInteger()** (p. ??).

## Parameters

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>nValue</i>	the value to assign.

References CPLStrdup(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OFTInteger, OFTIntegerList, OFTReal, OFTRealList, and OFTString.

Referenced by Clone(), OGRGenSQLResultsLayer::GetFeature(), SetField(), and SetFieldsFrom().

#### 12.50.3.25 void OGRFeature::SetField ( int *iField*, double *dfValue* )

Set field to double value.

OFTInteger and OFTReal fields will be set directly. OFTString fields will be assigned a string representation of the value, but not necessarily taking into account formatting constraints on this field. Other field types may be unaffected.

This method is the same as the C function **OGR\_F\_SetFieldDouble()** (p. ??).

## Parameters

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>dfValue</i>	the value to assign.

References CPLStrdup(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OFTInteger, OFTIntegerList, OFTReal, OFTRealList, OFTString, and SetField().

#### 12.50.3.26 void OGRFeature::SetField ( int *iField*, const char \* *pszValue* )

Set field to string value.

OFTInteger fields will be set based on an atoi() conversion of the string. OFTReal fields will be set based on an atof() conversion of the string. Other field types may be unaffected.

This method is the same as the C function **OGR\_F\_SetFieldString()** (p. ??).

## Parameters

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>pszValue</i>	the value to assign.

References CPLError(), CPLGetConfigOption(), CPLStrdup(), CPLStrtod(), CSLCount(), CSLDestroy(), CSLTestBoolean(), CSLTokenizeString2(), OGRFeatureDefn::GetFieldDefn(), OGRFeatureDefn::GetName(), OGRFieldDefn::GetNameRef(), OGRFieldDefn::GetType(), IsFieldSet(), OFTDate, OFTDateTime, OFTInteger, OFTIntegerList, OFTReal, OFTRealList, OFTString, OFTTime, OGRParseDate(), and SetField().

### 12.50.3.27 void OGRFeature::SetField ( int *iField*, int *nCount*, int \* *panValues* )

Set field to list of integers value.

This method currently on has an effect of OFTIntegerList fields.

This method is the same as the C function **OGR\_F\_SetFieldIntegerList()** (p. ??).

## Parameters

<i>iField</i>	the field to set, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>nCount</i>	the number of values in the list being assigned.
<i>panValues</i>	the values to assign.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTInteger, OFTIntegerList, OFTReal, OFTRealList, and SetField().

### 12.50.3.28 void OGRFeature::SetField ( int *iField*, int *nCount*, double \* *padfValues* )

Set field to list of doubles value.

This method currently on has an effect of OFTRealList fields.

This method is the same as the C function **OGR\_F\_SetFieldDoubleList()** (p. ??).

## Parameters

<i>iField</i>	the field to set, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>nCount</i>	the number of values in the list being assigned.
<i>padfValues</i>	the values to assign.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTInteger, OFTIntegerList, OFTReal, OFTRealList, and SetField().

### 12.50.3.29 void OGRFeature::SetField ( int *iField*, char \*\* *papszValues* )

Set field to list of strings value.

This method currently on has an effect of OFTStringList fields.

This method is the same as the C function **OGR\_F\_SetFieldStringList()** (p. ??).

## Parameters

<i>iField</i>	the field to set, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>papszValues</i>	the values to assign.

References CSLCount(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTStringList, and SetField().

### 12.50.3.30 void OGRFeature::SetField ( int *iField*, OGRField \* *puValue* )

Set field.

The passed value **OGRField** (p. ??) must be of exactly the same type as the target field, or an application crash may occur. The passed value is copied, and will not be affected. It remains the responsibility of the caller.

This method is the same as the C function **OGR\_F\_SetFieldRaw()** (p. ??).

#### Parameters

<i>iField</i>	the field to fetch, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>puValue</i>	the value to assign.

References CPLMalloc(), CPLStrdup(), CSLCount(), CSLDestroy(), CSLDuplicate(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OFTBinary, OFTDate, OFTDateTime, OFTInteger, OFTIntegerList, OFTReal, OFTRealList, OFTString, OFTStringList, and OFTTime.

#### 12.50.3.31 void OGRFeature::SetField ( int *iField*, int *nBytes*, GByte \* *pabyData* )

Set field to binary data.

This method currently on has an effect of OFTBinary fields.

This method is the same as the C function **OGR\_F\_SetFieldBinary()** (p. ??).

#### Parameters

<i>iField</i>	the field to set, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>nBytes</i>	bytes of data being set.
<i>pabyData</i>	the raw data being applied.

References OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTBinary, and SetField().

#### 12.50.3.32 void OGRFeature::SetField ( int *iField*, int *nYear*, int *nMonth*, int *nDay*, int *nHour* = 0, int *nMinute* = 0, int *nSecond* = 0, int *nTZFlag* = 0 )

Set field to date.

This method currently only has an effect for OFTDate, OFTTime and OFTDateTime fields.

This method is the same as the C function **OGR\_F\_SetFieldDateTime()** (p. ??).

#### Parameters

<i>iField</i>	the field to set, from 0 to <b>GetFieldCount()</b> (p. ??)-1.
<i>nYear</i>	(including century)
<i>nMonth</i>	(1-12)
<i>nDay</i>	(1-31)
<i>nHour</i>	(0-23)
<i>nMinute</i>	(0-59)
<i>nSecond</i>	(0-59)
<i>nTZFlag</i>	(0=unknown, 1=localtime, 100=GMT, see data model for details)

References CPLError(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), OFTDate, OFTDateTime, and OFTTime.

#### 12.50.3.33 OGRErr OGRFeature::SetFieldsFrom ( OGRFeature \* *poSrcFeature*, int \* *panMap*, int *bForgiving* = TRUE )

Set fields from another feature.

Overwrite the fields of this feature from the attributes of another. The FID and the style string are not set. The *poSrcFeature* does not need to have the same **OGRFeatureDefn** (p. ??). Field values are copied according to the provided indices map. Field types do not have to exactly match. **SetField()** (p. ??) method conversion rules will be applied as needed. This is more efficient than **OGR\_F\_SetFrom()** (p. ??) in that this doesn't lookup the fields by their names. Particularly useful when the field names don't match.



## Parameters

<i>poSrcFeature</i>	the feature from which geometry, and field values will be copied.
<i>panMap</i>	Array of the indices of the feature's fields stored at the corresponding index of the source feature's fields. A value of -1 should be used to ignore the source's field. The array should not be NULL and be as long as the number of fields in the source feature.
<i>bForgiving</i>	TRUE if the operation should continue despite lacking output fields matching some of the source fields.

## Returns

OGRERR\_NONE if the operation succeeds, even if some values are not transferred, otherwise an error code.

References `GetFieldAsDouble()`, `GetFieldAsDoubleList()`, `GetFieldAsInteger()`, `GetFieldAsIntegerList()`, `GetFieldAsString()`, `GetFieldCount()`, `GetFieldDefnRef()`, `GetRawFieldRef()`, `OGRFieldDefn::GetType()`, `IsFieldSet()`, `OFTDate`, `OFTDateTime`, `OFTInteger`, `OFTIntegerList`, `OFTReal`, `OFTRealList`, `OFTString`, `OFTTime`, `SetField()`, and `UnsetField()`.

Referenced by `OGRLayer::Clip()`, `OGRLayer::Erase()`, `OGRLayer::Identity()`, `OGRLayer::Intersection()`, `SetFrom()`, `OGRLayer::SymDifference()`, `OGRLayer::Union()`, and `OGRLayer::Update()`.

#### 12.50.3.34 OGRErr OGRFeature::SetFrom ( OGRFeature \* poSrcFeature, int bForgiving = TRUE )

Set one feature from another.

Overwrite the contents of this feature from the geometry and attributes of another. The `poSrcFeature` does not need to have the same **OGRFeatureDefn** (p. ??). Field values are copied by corresponding field names. Field types do not have to exactly match. **SetField()** (p. ??) method conversion rules will be applied as needed.

This method is the same as the C function **OGR\_F\_SetFrom()** (p. ??).

## Parameters

<i>poSrcFeature</i>	the feature from which geometry, and field values will be copied.
<i>bForgiving</i>	TRUE if the operation should continue despite lacking output fields matching some of the source fields.

## Returns

OGRERR\_NONE if the operation succeeds, even if some values are not transferred, otherwise an error code.

References `GetFieldCount()`, `GetFieldDefnRef()`, `GetFieldIndex()`, and `OGRFieldDefn::GetNameRef()`.

Referenced by `OGRDataSource::CopyLayer()`, `OGRUnionLayer::CreateFeature()`, and `OGRUnionLayer::SetFeature()`.

#### 12.50.3.35 OGRErr OGRFeature::SetFrom ( OGRFeature \* poSrcFeature, int \* panMap, int bForgiving = TRUE )

Set one feature from another.

Overwrite the contents of this feature from the geometry and attributes of another. The `poSrcFeature` does not need to have the same **OGRFeatureDefn** (p. ??). Field values are copied according to the provided indices map. Field types do not have to exactly match. **SetField()** (p. ??) method conversion rules will be applied as needed. This is more efficient than **OGR\_F\_SetFrom()** (p. ??) in that this doesn't lookup the fields by their names. Particularly useful when the field names don't match.

This method is the same as the C function **OGR\_F\_SetFromWithMap()** (p. ??).

## Parameters

<i>poSrcFeature</i>	the feature from which geometry, and field values will be copied.
<i>panMap</i>	Array of the indices of the feature's fields stored at the corresponding index of the source feature's fields. A value of -1 should be used to ignore the source's field. The array should not be NULL and be as long as the number of fields in the source feature.
<i>bForgiving</i>	TRUE if the operation should continue despite lacking output fields matching some of the source fields.

## Returns

OGRERR\_NONE if the operation succeeds, even if some values are not transferred, otherwise an error code.

References GetGeometryRef(), GetStyleString(), SetFID(), SetFieldsFrom(), SetGeometry(), and SetStyleString().

### 12.50.3.36 OGRErr OGRFeature::SetGeometry ( OGRGeometry \* poGeomIn )

Set feature geometry.

This method updates the features geometry, and operate exactly as **SetGeometryDirectly()** (p. ??), except that this method does not assume ownership of the passed geometry, but instead makes a copy of it.

This method is the same as the C function **OGR\_F\_SetGeometry()** (p. ??).

## Parameters

<i>poGeomIn</i>	new geometry to apply to feature. Passing NULL value here is correct and it will result in deallocation of currently assigned geometry without assigning new one.
-----------------	---

## Returns

OGRERR\_NONE if successful, or OGR\_UNSUPPORTED\_GEOMETRY\_TYPE if the geometry type is illegal for the **OGRFeatureDefn** (p. ??) (checking not yet implemented).

References OGRGeometry::clone().

Referenced by Clone(), SetFrom(), and OGRLayer::Update().

### 12.50.3.37 OGRErr OGRFeature::SetGeometryDirectly ( OGRGeometry \* poGeomIn )

Set feature geometry.

This method updates the features geometry, and operate exactly as **SetGeometry()** (p. ??), except that this method assumes ownership of the passed geometry.

This method is the same as the C function **OGR\_F\_SetGeometryDirectly()** (p. ??).

## Parameters

<i>poGeomIn</i>	new geometry to apply to feature. Passing NULL value here is correct and it will result in deallocation of currently assigned geometry without assigning new one.
-----------------	---

## Returns

OGRERR\_NONE if successful, or OGR\_UNSUPPORTED\_GEOMETRY\_TYPE if the geometry type is illegal for the **OGRFeatureDefn** (p. ??) (checking not yet implemented).

Referenced by OGRLayer::Clip(), OGRWarpedLayer::CreateFeature(), OGRLayer::Erase(), OGRLayer::Identity(), OGRLayer::Intersection(), OGRWarpedLayer::SetFeature(), OGRLayer::SymDifference(), OGRLayer::Union(), and OGRLayer::Update().

**12.50.3.38 void OGRFeature::SetStyleString ( const char \* *pszString* ) [virtual]**

Set feature style string. This method operate exactly as **OGRFeature::SetStyleStringDirectly()** (p. ??) except that it does not assume ownership of the passed string, but instead makes a copy of it.

This method is the same as the C function **OGR\_F\_SetStyleString()** (p. ??).

**Parameters**

<i>pszString</i>	the style string to apply to this feature, cannot be NULL.
------------------	--

References CPLStrdup().

Referenced by Clone(), OGRStyleMgr::SetFeatureStyleString(), and SetFrom().

**12.50.3.39 void OGRFeature::SetStyleStringDirectly ( char \* *pszString* ) [virtual]**

Set feature style string. This method operate exactly as **OGRFeature::SetStyleString()** (p. ??) except that it assumes ownership of the passed string.

This method is the same as the C function **OGR\_F\_SetStyleStringDirectly()** (p. ??).

**Parameters**

<i>pszString</i>	the style string to apply to this feature, cannot be NULL.
------------------	--

**12.50.3.40 OGRGeometry \* OGRFeature::StealGeometry ( )**

Take away ownership of geometry.

Fetch the geometry from this feature, and clear the reference to the geometry on the feature. This is a mechanism for the application to take over ownership of the geometry from the feature without copying. Sort of an inverse to **SetGeometryDirectly()** (p. ??).

After this call the **OGRFeature** (p. ??) will have a NULL geometry.

**Returns**

the pointer to the geometry.

Referenced by OGRWarpedLayer::CreateFeature(), OGRWarpedLayer::GetFeature(), OGRWarpedLayer::GetNextFeature(), and OGRWarpedLayer::SetFeature().

**12.50.3.41 void OGRFeature::UnsetField ( int *iField* )**

Clear a field, marking it as unset.

This method is the same as the C function **OGR\_F\_UnsetField()** (p. ??).

**Parameters**

<i>iField</i>	the field to unset.
---------------	---------------------

References CSLDestroy(), OGRFeatureDefn::GetFieldDefn(), OGRFieldDefn::GetType(), IsFieldSet(), OFTBinary, OFTIntegerList, OFTRealList, OFTString, and OFTStringList.

Referenced by OGRGenSQLResultsLayer::GetFeature(), and SetFieldsFrom().

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

- **ogr\_feature.h**
- **ogrfeature.cpp**

## 12.51 OGRFeatureDefn Class Reference

```
#include <ogr_feature.h>
```

### Public Member Functions

- **OGRFeatureDefn** (const char \*pszName=NULL)  
*Constructor.*
- const char \* **GetName** ()  
*Get name of this **OGRFeatureDefn** (p. ??).*
- int **GetFieldCount** ()  
*Fetch number of fields on this feature.*
- **OGRFieldDefn** \* **GetFieldDefn** (int i)  
*Fetch field definition.*
- int **GetFieldIndex** (const char \*)  
*Find field by name.*
- void **AddFieldDefn** (**OGRFieldDefn** \*)  
*Add a new field definition.*
- OGRErr **DeleteFieldDefn** (int iField)  
*Delete an existing field definition.*
- OGRErr **ReorderFieldDefns** (int \*panMap)  
*Reorder the field definitions in the array of the feature definition.*
- **OGRwkbGeometryType** **GetGeomType** ()  
*Fetch the geometry base type.*
- void **SetGeomType** (**OGRwkbGeometryType**)  
*Assign the base geometry type for this layer.*
- **OGRFeatureDefn** \* **Clone** ()  
*Create a copy of this feature definition.*
- int **Reference** ()  
*Increments the reference count by one.*
- int **Dereference** ()  
*Decrements the reference count by one.*
- int **GetReferenceCount** ()  
*Fetch current reference count.*
- void **Release** ()  
*Drop a reference to this object, and destroy if no longer referenced.*
- int **IsGeometryIgnored** ()  
*Determine whether the geometry can be omitted when fetching features.*
- void **SetGeometryIgnored** (int blgnore)  
*Set whether the geometry can be omitted when fetching features.*
- int **IsStyleIgnored** ()  
*Determine whether the style can be omitted when fetching features.*
- void **SetStyleIgnored** (int blgnore)  
*Set whether the style can be omitted when fetching features.*
- int **IsSame** (const **OGRFeatureDefn** \*poOtherFeatureDefn) const  
*Test if the feature definition is identical to the other one.*

### 12.51.1 Detailed Description

Definition of a feature class or feature layer.

This object contains schema information for a set of OGRFeatures. In table based systems, an **OGRFeatureDefn** (p. ??) is essentially a layer. In more object oriented approaches (such as SF CORBA) this can represent a class of features but doesn't necessarily relate to all of a layer, or just one layer.

This object also can contain some other information such as a name, the base geometry type and potentially other metadata.

It is reasonable for different translators to derive classes from **OGRFeatureDefn** (p. ??) with additional translator specific information.

### 12.51.2 Constructor & Destructor Documentation

#### 12.51.2.1 OGRFeatureDefn::OGRFeatureDefn ( const char \* *pszName* = NULL )

Constructor.

The **OGRFeatureDefn** (p. ??) maintains a reference count, but this starts at zero. It is mainly intended to represent a count of **OGRFeature** (p. ??)'s based on this definition.

This method is the same as the C function **OGR\_FD\_Create()** (p. ??).

Parameters

<i>pszName</i>	the name to be assigned to this layer/class. It does not need to be unique.
----------------	---

References CPLStrdup(), and wkbUnknown.

Referenced by Clone().

### 12.51.3 Member Function Documentation

#### 12.51.3.1 void OGRFeatureDefn::AddFieldDefn ( OGRFieldDefn \* *poNewDefn* )

Add a new field definition.

To add a new field definition to a layer definition, do not use this function directly, but use **OGRLayer::CreateField()** (p. ??) instead.

This method should only be called while there are no **OGRFeature** (p. ??) objects in existence based on this **OGRFeatureDefn** (p. ??). The **OGRFieldDefn** (p. ??) passed in is copied, and remains the responsibility of the caller.

This method is the same as the C function **OGR\_FD\_AddFieldDefn()** (p. ??).

Parameters

<i>poNewDefn</i>	the definition of the new field.
------------------	----------------------------------

References CPLRealloc().

Referenced by Clone(), and OGRUnionLayer::GetLayerDefn().

#### 12.51.3.2 OGRFeatureDefn \* OGRFeatureDefn::Clone ( )

Create a copy of this feature definition.

Creates a deep copy of the feature definition.

**Returns**

the copy.

References AddFieldDefn(), GetFieldCount(), GetFieldDefn(), GetGeomType(), GetName(), OGRFeatureDefn(), and SetGeomType().

**12.51.3.3 OGRErr OGRFeatureDefn::DeleteFieldDefn ( int *iField* )**

Delete an existing field definition.

To delete an existing field definition from a layer definition, do not use this function directly, but use **OGRLayer::DeleteField()** (p. ??) instead.

This method should only be called while there are no **OGRFeature** (p. ??) objects in existence based on this **OGRFeatureDefn** (p. ??).

This method is the same as the C function **OGR\_FD\_DeleteFieldDefn()** (p. ??).

**Parameters**

<i>iField</i>	the index of the field definition.
---------------	------------------------------------

**Returns**

OGRERR\_NONE in case of success.

**Since**

OGR 1.9.0

Referenced by OGRUnionLayer::GetLayerDefn().

**12.51.3.4 int OGRFeatureDefn::Dereference ( ) [inline]**

Decrements the reference count by one.

This method is the same as the C function **OGR\_FD\_Dereference()** (p. ??).

**Returns**

the updated reference count.

Referenced by Release().

**12.51.3.5 int OGRFeatureDefn::GetFieldCount ( ) [inline]**

Fetch number of fields on this feature.

This method is the same as the C function **OGR\_FD\_GetFieldCount()** (p. ??).

**Returns**

count of fields.

Referenced by Clone(), OGRFeature::Clone(), OGRDataSource::CopyLayer(), OGRFeature::Equal(), OGRFeature::GetFieldAsDouble(), OGRFeature::GetFieldAsInteger(), OGRFeature::GetFieldAsString(), OGRUnionLayer::GetLayerDefn(), OGRFeature::IsFieldSet(), OGRFeature::OGRFeature(), OGRLayer::ReorderField(), and OGRLayer::SetIgnoredFields().

### 12.51.3.6 OGRFieldDefn \* OGRFeatureDefn::GetFieldDefn ( int *iField* )

Fetch field definition.

This method is the same as the C function **OGR\_FD\_GetFieldDefn()** (p. ??).

Starting with GDAL 1.7.0, this method will also issue an error if the index is not valid.

#### Parameters

<i>iField</i>	the field to fetch, between 0 and <b>GetFieldCount()</b> (p. ??)-1.
---------------	---

#### Returns

a pointer to an internal field definition object or NULL if invalid index. This object should not be modified or freed by the application.

References CPLError().

Referenced by Clone(), OGRDataSource::CopyLayer(), OGRFeature::DumpReadable(), OGRFeature::Equal(), OGRFeature::GetFieldAsBinary(), OGRFeature::GetFieldAsDateTime(), OGRFeature::GetFieldAsDouble(), OGRFeature::GetFieldAsDoubleList(), OGRFeature::GetFieldAsInteger(), OGRFeature::GetFieldAsIntegerList(), OGRFeature::GetFieldAsString(), OGRFeature::GetFieldAsStringList(), OGRUnionLayer::GetLayerDefn(), OGRFeature::SetField(), OGRLayer::SetIgnoredFields(), and OGRFeature::UnsetField().

### 12.51.3.7 int OGRFeatureDefn::GetFieldIndex ( const char \* *pszFieldName* )

Find field by name.

The field index of the first field matching the passed field name (case insensitively) is returned.

This method is the same as the C function **OGR\_FD\_GetFieldIndex()** (p. ??).

#### Parameters

<i>pszFieldName</i>	the field name to search for.
---------------------	-------------------------------

#### Returns

the field index, or -1 if no match found.

Referenced by OGRDataSource::CopyLayer(), OGRUnionLayer::GetLayerDefn(), and OGRLayer::SetIgnoredFields().

### 12.51.3.8 OGRwkbGeometryType OGRFeatureDefn::GetGeomType ( ) [inline]

Fetch the geometry base type.

Note that some drivers are unable to determine a specific geometry type for a layer, in which case wkbUnknown is returned. A value of wkbNone indicates no geometry is available for the layer at all. Many drivers do not properly mark the geometry type as 25D even if some or all geometries are in fact 25D. A few (broken) drivers return wkbPolygon for layers that also include wkbMultiPolygon.

This method is the same as the C function **OGR\_FD\_GetGeomType()** (p. ??).

#### Returns

the base type for all geometry related to this definition.

Referenced by Clone(), OGRDataSource::CopyLayer(), and OGRLayer::GetGeomType().

### 12.51.3.9 `const char * OGRFeatureDefn::GetName ( ) [inline]`

Get name of this **OGRFeatureDefn** (p. ??).

This method is the same as the C function **OGR\_FD\_GetName()** (p. ??).

#### Returns

the name. This name is internal and should not be modified, or freed.

Referenced by `Clone()`, `OGRSFDriver::CopyDataSource()`, `OGRDataSource::CopyLayer()`, `OGRFeature::DumpReadable()`, `OGRLayer::GetName()`, and `OGRFeature::SetField()`.

### 12.51.3.10 `int OGRFeatureDefn::GetReferenceCount ( ) [inline]`

Fetch current reference count.

This method is the same as the C function **OGR\_FD\_GetReferenceCount()** (p. ??).

#### Returns

the current reference count.

### 12.51.3.11 `int OGRFeatureDefn::IsGeometryIgnored ( ) [inline]`

Determine whether the geometry can be omitted when fetching features.

This method is the same as the C function **OGR\_FD\_IsGeometryIgnored()** (p. ??).

#### Returns

ignore state

### 12.51.3.12 `int OGRFeatureDefn::IsSame ( const OGRFeatureDefn * poOtherFeatureDefn ) const`

Test if the feature definition is identical to the other one.

#### Parameters

<i>poOtherFeatureDefn</i>	the other feature definition to compare to.
---------------------------	---

#### Returns

TRUE if the feature definition is identical to the other one.

References `OGRFieldDefn::IsSame()`.

### 12.51.3.13 `int OGRFeatureDefn::IsStyleIgnored ( ) [inline]`

Determine whether the style can be omitted when fetching features.

This method is the same as the C function **OGR\_FD\_IsStyleIgnored()** (p. ??).

#### Returns

ignore state



#### 12.51.3.14 int OGRFeatureDefn::Reference ( ) [inline]

Increments the reference count by one.

The reference count is used keep track of the number of **OGRFeature** (p. ??) objects referencing this definition.

This method is the same as the C function **OGR\_FD\_Reference()** (p. ??).

##### Returns

the updated reference count.

Referenced by OGRProxiedLayer::GetLayerDefn(), OGRUnionLayer::GetLayerDefn(), and OGRFeature::OGRFeature().

#### 12.51.3.15 OGRErr OGRFeatureDefn::ReorderFieldDefns ( int \* panMap )

Reorder the field definitions in the array of the feature definition.

To reorder the field definitions in a layer definition, do not use this function directly, but use **OGR\_L\_ReorderFields()** (p. ??) instead.

This method should only be called while there are no **OGRFeature** (p. ??) objects in existence based on this **OGRFeatureDefn** (p. ??).

This method is the same as the C function OGR\_FD\_ReorderFieldDefns().

##### Parameters

<i>panMap</i>	an array of <b>GetFieldCount()</b> (p. ??) elements which is a permutation of [0, <b>GetFieldCount()</b> (p. ??)-1]. <i>panMap</i> is such that, for each field definition at position <i>i</i> after reordering, its position before reordering was <i>panMap[i]</i> .
---------------	---

##### Returns

OGRERR\_NONE in case of success.

##### Since

OGR 1.9.0

References CPLMalloc().

#### 12.51.3.16 void OGRFeatureDefn::SetGeometryIgnored ( int *blgnore* ) [inline]

Set whether the geometry can be omitted when fetching features.

This method is the same as the C function **OGR\_FD\_SetGeometryIgnored()** (p. ??).

##### Parameters

<i>blgnore</i>	ignore state
----------------	--------------

Referenced by OGRLayer::SetIgnoredFields().

#### 12.51.3.17 void OGRFeatureDefn::SetGeomType ( OGRwkbGeometryType *eNewType* )

Assign the base geometry type for this layer.

All geometry objects using this type must be of the defined type or a derived type. The default upon creation is wkbUnknown which allows for any geometry type. The geometry type should generally not be changed after any OGRFeatures have been created against this definition.

This method is the same as the C function **OGR\_FD\_SetGeomType()** (p. ??).

## Parameters

<i>eNewType</i>	the new type to assign.
-----------------	-------------------------

Referenced by Clone(), and OGRUnionLayer::GetLayerDefn().

12.51.3.18 void OGRFeatureDefn::SetStyleIgnored ( int *blgnore* ) [inline]

Set whether the style can be omitted when fetching features.

This method is the same as the C function **OGR\_FD\_SetStyleIgnored()** (p. ??).

## Parameters

<i>blgnore</i>	ignore state
----------------	--------------

Referenced by OGRLayer::SetIgnoredFields().

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

- **ogr\_feature.h**
- ogrfeaturedefn.cpp

## 12.52 OGRFeatureQuery Class Reference

### Public Member Functions

- char \*\* **GetUsedFields** ()

#### 12.52.1 Member Function Documentation

12.52.1.1 char \*\* OGRFeatureQuery::GetUsedFields ( )

Returns lists of fields in expression.

All attribute fields are used in the expression of this feature query are returned as a StringList of field names. This function would primarily be used within drivers to recognise special case conditions depending only on attribute fields that can be very efficiently fetched.

NOTE: If any fields in the expression are from tables other than the primary table then NULL is returned indicating an error. In succesful use, no non-empty expression should return an empty list.

## Returns

list of field names. Free list with **CSLDestroy()** (p. ??) when no longer required.

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

- **ogr\_feature.h**
- ogrfeaturequery.cpp

## 12.53 OGRField Union Reference

```
#include <ogr_core.h>
```

### 12.53.1 Detailed Description

**OGRFeature** (p. ??) field attribute value union.

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

- **ogr\_core.h**

## 12.54 OGRFieldDefn Class Reference

```
#include <ogr_feature.h>
```

### Public Member Functions

- **OGRFieldDefn** (const char \*, **OGRFieldType**)  
*Constructor.*
- **OGRFieldDefn** (**OGRFieldDefn** \*)  
*Constructor.*
- void **SetName** (const char \*)  
*Reset the name of this field.*
- const char \* **GetNameRef** ()  
*Fetch name of this field.*
- **OGRFieldType** **GetType** ()  
*Fetch type of this field.*
- void **SetType** (**OGRFieldType** eTypeIn)  
*Set the type of this field. This should never be done to an **OGRFieldDefn** (p. ??) that is already part of an **OGR-FeatureDefn** (p. ??).*
- **OGRJustification** **GetJustify** ()  
*Get the justification for this field.*
- void **SetJustify** (**OGRJustification** eJustifyIn)  
*Set the justification for this field.*
- int **GetWidth** ()  
*Get the formatting width for this field.*
- void **SetWidth** (int nWidthIn)  
*Set the formatting width for this field in characters.*
- int **GetPrecision** ()  
*Get the formatting precision for this field. This should normally be zero for fields of types other than OFTReal.*
- void **SetPrecision** (int nPrecisionIn)  
*Set the formatting precision for this field in characters.*
- void **Set** (const char \*, **OGRFieldType**, int=0, int=0, **OGRJustification**=OJUUndefined)  
*Set defining parameters for a field in one call.*
- void **SetDefault** (const **OGRField** \*)  
*Set default field value.*
- int **IsIgnored** ()  
*Return whether this field should be omitted when fetching features.*
- void **SetIgnored** (int bIgnore)  
*Set whether this field should be omitted when fetching features.*
- int **IsSame** (const **OGRFieldDefn** \*) const  
*Test if the field definition is identical to the other one.*

## Static Public Member Functions

- static const char \* **GetFieldTypeName** (OGRFieldType)
- Fetch human readable name for a field type.*

### 12.54.1 Detailed Description

Definition of an attribute of an **OGRFeatureDefn** (p. ??).

### 12.54.2 Constructor & Destructor Documentation

#### 12.54.2.1 OGRFieldDefn::OGRFieldDefn ( const char \* pszNameIn, OGRFieldType eTypeIn )

Constructor.

Parameters

<i>pszNameIn</i>	the name of the new field.
<i>eTypeIn</i>	the type of the new field.

#### 12.54.2.2 OGRFieldDefn::OGRFieldDefn ( OGRFieldDefn \* poPrototype )

Constructor.

Create by cloning an existing field definition.

Parameters

<i>poPrototype</i>	the field definition to clone.
--------------------	--------------------------------

References GetJustify(), GetNameRef(), GetPrecision(), GetType(), GetWidth(), SetJustify(), SetPrecision(), and SetWidth().

### 12.54.3 Member Function Documentation

#### 12.54.3.1 const char \* OGRFieldDefn::GetFieldTypeName ( OGRFieldType eType ) [static]

Fetch human readable name for a field type.

This static method is the same as the C function **OGR\_GetFieldTypeName()** (p. ??).

Parameters

<i>eType</i>	the field type to get name for.
--------------	---------------------------------

Returns

pointer to an internal static name string. It should not be modified or freed.

References OFTBinary, OFTDate, OFTDateTime, OFTInteger, OFTIntegerList, OFTReal, OFTRealList, OFTString, OFTStringList, and OFTTime.

Referenced by OGRFeature::DumpReadable(), and OGR\_GetFieldTypeName().

#### 12.54.3.2 OGRJustification OGRFieldDefn::GetJustify ( ) [inline]

Get the justification for this field.

This method is the same as the C function **OGR\_Fld\_GetJustify()** (p. ??).

**Returns**

the justification.

Referenced by OGRFieldDefn().

**12.54.3.3 const char \* OGRFieldDefn::GetNameRef ( ) [inline]**

Fetch name of this field.

This method is the same as the C function **OGR\_Fld\_GetNameRef()** (p. ??).

**Returns**

pointer to an internal name string that should not be freed or modified.

Referenced by OGRDataSource::CopyLayer(), OGRFeature::DumpReadable(), OGRUnionLayer::GetLayerDefn(), OGRFieldDefn(), OGRFeature::SetField(), and OGRFeature::SetFrom().

**12.54.3.4 int OGRFieldDefn::GetPrecision ( ) [inline]**

Get the formatting precision for this field. This should normally be zero for fields of types other than OFTReal.

This method is the same as the C function **OGR\_Fld\_GetPrecision()** (p. ??).

**Returns**

the precision.

Referenced by OGRFeature::GetFieldAsString(), and OGRFieldDefn().

**12.54.3.5 OGRFieldType OGRFieldDefn::GetType ( ) [inline]**

Fetch type of this field.

This method is the same as the C function **OGR\_Fld\_GetType()** (p. ??).

**Returns**

field type.

Referenced by OGRFeature::DumpReadable(), OGRFeature::Equal(), OGRFeature::GetFieldAsBinary(), OGRFeature::GetFieldAsDateTime(), OGRFeature::GetFieldAsDouble(), OGRFeature::GetFieldAsDoubleList(), OGRFeature::GetFieldAsInteger(), OGRFeature::GetFieldAsIntegerList(), OGRFeature::GetFieldAsString(), OGRFeature::GetFieldAsStringList(), OGRFieldDefn(), OGRFeature::SetField(), OGRFeature::SetFieldsFrom(), and OGRFeature::UnsetField().

**12.54.3.6 int OGRFieldDefn::GetWidth ( ) [inline]**

Get the formatting width for this field.

This method is the same as the C function **OGR\_Fld\_GetWidth()** (p. ??).

**Returns**

the width, zero means no specified width.

Referenced by OGRFeature::GetFieldAsString(), and OGRFieldDefn().

#### 12.54.3.7 int OGRFieldDefn::IsIgnored ( ) [inline]

Return whether this field should be omitted when fetching features.

This method is the same as the C function **OGR\_Fld\_IsIgnored()** (p. ??).

##### Returns

ignore state

#### 12.54.3.8 int OGRFieldDefn::IsSame ( const OGRFieldDefn \* poOtherFieldDefn ) const

Test if the field definition is identical to the other one.

##### Parameters

<i>poOtherFieldDefn</i>	the other field definition to compare to.
-------------------------	---

##### Returns

TRUE if the field definition is identical to the other one.

Referenced by OGRFeatureDefn::IsSame().

#### 12.54.3.9 void OGRFieldDefn::Set ( const char \* pszNameIn, OGRFieldType eTypeIn, int nWidthIn = 0, int nPrecisionIn = 0, OGRJustification eJustifyIn = OJUndefined )

Set defining parameters for a field in one call.

This method is the same as the C function **OGR\_Fld\_Set()** (p. ??).

##### Parameters

<i>pszNameIn</i>	the new name to assign.
<i>eTypeIn</i>	the new type (one of the OFT values like OFTInteger).
<i>nWidthIn</i>	the preferred formatting width. Defaults to zero indicating undefined.
<i>nPrecisionIn</i>	number of decimals places for formatting, defaults to zero indicating undefined.
<i>eJustifyIn</i>	the formatting justification (OJLeft or OJRight), defaults to OJUndefined.

References SetJustify(), SetName(), SetPrecision(), SetType(), and SetWidth().

#### 12.54.3.10 void OGRFieldDefn::SetDefault ( const OGRField \* puDefaultIn )

Set default field value.

Currently use of **OGRFieldDefn** (p. ??) "defaults" is discouraged. This feature may be fleshed out in the future.

References OFTInteger, OFTReal, and OFTString.

#### 12.54.3.11 void OGRFieldDefn::SetIgnored ( int ignore ) [inline]

Set whether this field should be omitted when fetching features.

This method is the same as the C function **OGR\_Fld\_SetIgnored()** (p. ??).

## Parameters

<i>ignore</i>	ignore state
---------------	--------------

Referenced by OGRLayer::SetIgnoredFields().

#### 12.54.3.12 void OGRFieldDefn::SetJustify ( OGRJustification *eJustify* ) [inline]

Set the justification for this field.

This method is the same as the C function **OGR\_Fld\_SetJustify()** (p. ??).

## Parameters

<i>eJustify</i>	the new justification.
-----------------	------------------------

Referenced by OGRFieldDefn(), and Set().

#### 12.54.3.13 void OGRFieldDefn::SetName ( const char \* *pszNameIn* )

Reset the name of this field.

This method is the same as the C function **OGR\_Fld\_SetName()** (p. ??).

## Parameters

<i>pszNameIn</i>	the new name to apply.
------------------	------------------------

References CPLStrdup().

Referenced by Set().

#### 12.54.3.14 void OGRFieldDefn::SetPrecision ( int *nPrecision* ) [inline]

Set the formatting precision for this field in characters.

This should normally be zero for fields of types other than OFTReal.

This method is the same as the C function **OGR\_Fld\_SetPrecision()** (p. ??).

## Parameters

<i>nPrecision</i>	the new precision.
-------------------	--------------------

Referenced by OGRFieldDefn(), and Set().

#### 12.54.3.15 void OGRFieldDefn::SetType ( OGRFieldType *eType* ) [inline]

Set the type of this field. This should never be done to an **OGRFieldDefn** (p. ??) that is already part of an **OGR-FeatureDefn** (p. ??).

This method is the same as the C function **OGR\_Fld\_SetType()** (p. ??).

## Parameters

<i>eType</i>	the new field type.
--------------	---------------------

Referenced by Set().

#### 12.54.3.16 void OGRFieldDefn::SetWidth ( int *nWidth* ) [inline]

Set the formatting width for this field in characters.

This method is the same as the C function **OGR\_Fld\_SetWidth()** (p. ??).

## Parameters

<i>nWidth</i>	the new width.
---------------	----------------

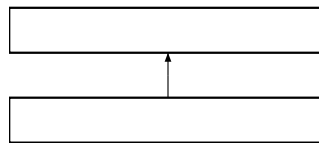
Referenced by OGRFieldDefn(), and Set().

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

- **ogr\_feature.h**
- ogrfielddefn.cpp

## 12.55 OGRGenSQLResultsLayer Class Reference

Inheritance diagram for OGRGenSQLResultsLayer:



### Public Member Functions

- virtual **OGRGeometry \*** **GetSpatialFilter** ()  
*This method returns the current spatial filter for this layer.*
- virtual void **ResetReading** ()  
*Reset feature reading to start on the first feature.*
- virtual **OGRFeature \*** **GetNextFeature** ()  
*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 **OGRFeatureDefn \*** **GetLayerDefn** ()  
*Fetch the schema information for this 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 int **TestCapability** (const char \*)  
*Test if this layer supported the named capability.*
- virtual void **SetSpatialFilter** (**OGRGeometry \***)  
*Set a new spatial filter.*
- virtual OGRErr **SetAttributeFilter** (const char \*)  
*Set a new attribute query.*



### 12.55.1 Member Function Documentation

#### 12.55.1.1 OGRErr OGRGenSQLResultsLayer::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()** (p. ??) 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()** (p. ??).

##### Parameters

<i>psExtent</i>	the structure in which the extent value will be returned.
<i>bForce</i>	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 from **OGRLayer** (p. ??).

References OGRLayer::GetExtent().

#### 12.55.1.2 OGRFeature \* OGRGenSQLResultsLayer::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.

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

Use OGRLayer::TestCapability(OLCRandomRead) to establish if this layer supports efficient random access reading via **GetFeature()** (p. ??); 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 are generally considered interrupted by a **GetFeature()** (p. ??) call.

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

This method is the same as the C function **OGR\_L\_GetFeature()** (p. ??).

##### Parameters

<i>nFID</i>	the feature id of the feature to read.
-------------	--

##### Returns

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

Reimplemented from **OGRLayer** (p. ??).

References OGRFeature::Clone(), OGRLayer::GetFeature(), OGRFeature::SetFID(), OGRFeature::SetField(), and OGRFeature::UnsetField().

Referenced by GetNextFeature().

### 12.55.1.3 `int OGRGenSQLResultsLayer::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()` (p. ??).

#### Parameters

<i>bForce</i>	Flag indicating whether the count should be computed even if it is expensive.
---------------	---

#### Returns

feature count, -1 if count not known.

Reimplemented from `OGRLayer` (p. ??).

References `OGRLayer::GetFeatureCount()`.

### 12.55.1.4 `OGRFeatureDefn * OGRGenSQLResultsLayer::GetLayerDefn ( ) [virtual]`

Fetch the schema information for this layer.

The returned `OGRFeatureDefn` (p. ??) is owned by the `OGRLayer` (p. ??), 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()` (p. ??).

#### Returns

feature definition.

Implements `OGRLayer` (p. ??).

### 12.55.1.5 `OGRFeature * OGRGenSQLResultsLayer::GetNextFeature ( ) [virtual]`

Fetch the next available feature from this layer.

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

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

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

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

#### Returns

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

Implements `OGRLayer` (p. ??).

References `GetFeature()`, and `OGRLayer::GetNextFeature()`.

**12.55.1.6 OGRGeometry \* OGRGenSQLResultsLayer::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()** (p. ??).

**Returns**

spatial filter geometry.

Reimplemented from **OGRLayer** (p. ??).

**12.55.1.7 OGRSpatialReference \* OGRGenSQLResultsLayer::GetSpatialRef ( ) [virtual]**

Fetch the spatial reference system for this layer.

The returned object is owned by the **OGRLayer** (p. ??) and should not be modified or freed by the application.

This method is the same as the C function **OGR\_L\_GetSpatialRef()** (p. ??).

**Returns**

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

Reimplemented from **OGRLayer** (p. ??).

References **OGRLayer::GetSpatialRef()**.

**12.55.1.8 void OGRGenSQLResultsLayer::ResetReading ( ) [virtual]**

Reset feature reading to start on the first feature.

This affects **GetNextFeature()** (p. ??).

This method is the same as the C function **OGR\_L\_ResetReading()** (p. ??).

Implements **OGRLayer** (p. ??).

References **OGRLayer::ResetReading()**, **OGRLayer::SetAttributeFilter()**, and **OGRLayer::SetSpatialFilter()**.

**12.55.1.9 OGRErr OGRGenSQLResultsLayer::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()** (p. ??) 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()** (p. ??)).

This method is the same as the C function **OGR\_L\_SetAttributeFilter()** (p. ??).

## Parameters

<i>pszQuery</i>	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 from **OGRLayer** (p. ??).

References OGRLayer::SetAttributeFilter().

#### 12.55.1.10 OGRErr OGRGenSQLResultsLayer::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()** (p. ??) 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()** (p. ??) 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()** (p. ??) efficiently implemented. In all other cases the default implementation which calls **ResetReading()** (p. ??) and then calls **GetNextFeature()** (p. ??) nIndex times is used. To determine if fast seeking is available on the current layer use the **TestCapability()** (p. ??) method with a value of OLCFastSetNextByIndex.

This method is the same as the C function **OGR\_L\_SetNextByIndex()** (p. ??).

## Parameters

<i>nIndex</i>	the index indicating how many steps into the result set to seek.
---------------	--

## Returns

OGRERR\_NONE on success or an error code.

Reimplemented from **OGRLayer** (p. ??).

References OGRLayer::SetNextByIndex().

#### 12.55.1.11 void OGRGenSQLResultsLayer::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()** (p. ??) 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()** (p. ??)) 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()** (p. ??)). In the future this may be generalized.

This method is the same as the C function **OGR\_L\_SetSpatialFilter()** (p. ??).

## Parameters

<i>poFilter</i>	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 from **OGRLayer** (p. ??).

References **OGRLayer::SetSpatialFilter()**.

#### 12.55.1.12 int OGRGenSQLResultsLayer::TestCapability ( const char \* *pszCap* ) [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()** (p. ??) method is implemented in an optimized way for this layer, as opposed to the default implementation using **ResetReading()** (p. ??) and **GetNextFeature()** (p. ??) to find the requested feature id.
- **OLCSequentialWrite** / "SequentialWrite": TRUE if the **CreateFeature()** (p. ??) method works for this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. ??) class may returned FALSE for other layer instances that are effectively read-only.
- **OLCRandomWrite** / "RandomWrite": TRUE if the **SetFeature()** (p. ??) method is operational on this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. ??) 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** (p. ??) 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()** (p. ??)) 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()** (p. ??)) 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()** (p. ??) call efficiently, otherwise FALSE.
- **OLCCreateField** / "CreateField": TRUE if this layer can create new fields on the current layer using **CreateField()** (p. ??), otherwise FALSE.
- **OLCDeleteField** / "DeleteField": TRUE if this layer can delete existing fields on the current layer using **DeleteField()** (p. ??), otherwise FALSE.
- **OLCReorderFields** / "ReorderFields": TRUE if this layer can reorder existing fields on the current layer using **ReorderField()** (p. ??) or **ReorderFields()** (p. ??), otherwise FALSE.
- **OLCAAlterFieldDefn** / "AlterFieldDefn": TRUE if this layer can alter the definition of an existing field on the current layer using **AlterFieldDefn()** (p. ??), otherwise FALSE.
- **OLCDeleteFeature** / "DeleteFeature": TRUE if the **DeleteFeature()** (p. ??) 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() (p. ??) method.

This method is the same as the C function **OGR\_L\_TestCapability()** (p. ??).

#### Parameters

<i>pszCap</i>	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.

Implements **OGRLayer** (p. ??).

References OGRLayer::TestCapability().

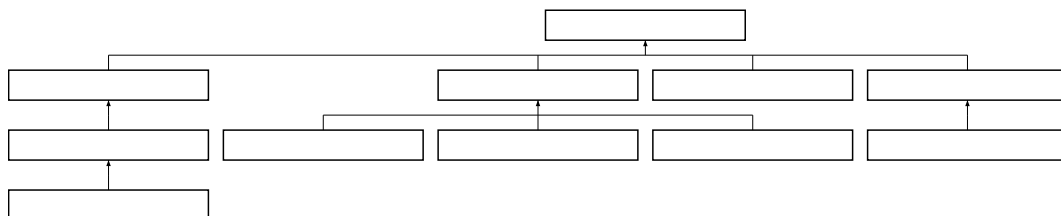
The documentation for this class was generated from the following files:

- ogr\_gensql.h
- ogr\_gensql.cpp

## 12.56 OGRGeometry Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRGeometry:



#### Public Member Functions

- virtual int **getDimension** () const =0  
*Get the dimension of this object.*
- virtual int **getCoordinateDimension** () const  
*Get the dimension of the coordinates in this object.*
- virtual OGRBoolean **IsEmpty** () const =0  
*Returns TRUE (non-zero) if the object has no points.*
- virtual OGRBoolean **IsValid** () const  
*Test if the geometry is valid.*
- virtual OGRBoolean **IsSimple** () const  
*Test if the geometry is simple.*
- virtual OGRBoolean **IsRing** () const  
*Test if the geometry is a ring.*
- virtual void **empty** ()=0

Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.

- virtual **OGRGeometry** \* **clone** () const =0

Make a copy of this object.

- virtual void **getEnvelope** (**OGREnvelope** \*psEnvelope) const =0

Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.

- virtual void **getEnvelope** (**OGREnvelope3D** \*psEnvelope) const =0

Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.

- virtual int **WkbSize** () const =0

Returns size of related binary representation.

- virtual **OGRERR** **importFromWkb** (unsigned char \*, int=-1)=0

Assign geometry from well known binary data.

- virtual **OGRERR** **exportToWkb** (**OGRwkbByteOrder**, unsigned char \*) const =0

Convert a geometry into well known binary format.

- virtual **OGRERR** **importFromWkt** (char \*\*ppszInput)=0

Assign geometry from well known text data.

- virtual **OGRERR** **exportToWkt** (char \*\*ppszDstText) const =0

Convert a geometry into well known text format.

- virtual **OGRwkbGeometryType** **getGeometryType** () const =0

Fetch geometry type.

- virtual const char \* **getGeometryName** () const =0

Fetch WKT name for geometry type.

- virtual void **dumpReadable** (FILE \*, const char \*\*=NULL, char \*\*ppszOptions=NULL) const

Dump geometry in well known text format to indicated output file.

- virtual void **flattenTo2D** ()=0

Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.

- virtual char \* **exportToGML** (const char \*const \*ppszOptions=NULL) const

Convert a geometry into GML format.

- virtual char \* **exportToKML** () const

Convert a geometry into KML format.

- virtual char \* **exportToJson** () const

Convert a geometry into GeoJSON format.

- virtual void **closeRings** ()

Force rings to be closed.

- virtual void **setCoordinateDimension** (int nDimension)

Set the coordinate dimension.

- void **assignSpatialReference** (**OGRSpatialReference** \*poSR)

Assign spatial reference to this object.

- **OGRSpatialReference** \* **getSpatialReference** (void) const

Returns spatial reference system for object.

- virtual **OGRERR** **transform** (**OGRCoordinateTransformation** \*poCT)=0

Apply arbitrary coordinate transformation to geometry.

- **OGRERR** **transformTo** (**OGRSpatialReference** \*poSR)

Transform geometry to new spatial reference system.

- virtual void **segmentize** (double dfMaxLength)

Modify the geometry such it has no segment longer then the given distance.

- virtual **OGRBoolean** **Intersects** (**OGRGeometry** \*) const

Do these features intersect?

- virtual **OGRBoolean** **Equals** (**OGRGeometry** \*) const =0

Returns TRUE if two geometries are equivalent.

- virtual **OGRBoolean** **Disjoint** (const **OGRGeometry** \*) const

- *Test for disjointness.*
- virtual OGRBoolean **Touches** (const **OGRGeometry** \*) const
- *Test for touching.*
- virtual OGRBoolean **Crosses** (const **OGRGeometry** \*) const
- *Test for crossing.*
- virtual OGRBoolean **Within** (const **OGRGeometry** \*) const
- *Test for containment.*
- virtual OGRBoolean **Contains** (const **OGRGeometry** \*) const
- *Test for containment.*
- virtual OGRBoolean **Overlaps** (const **OGRGeometry** \*) const
- *Test for overlap.*
- virtual **OGRGeometry** \* **Boundary** () const
- *Compute boundary.*
- virtual double **Distance** (const **OGRGeometry** \*) const
- *Compute distance between two geometries.*
- virtual **OGRGeometry** \* **ConvexHull** () const
- *Compute convex hull.*
- virtual **OGRGeometry** \* **Buffer** (double dfDist, int nQuadSegs=30) const
- *Compute buffer of geometry.*
- virtual **OGRGeometry** \* **Intersection** (const **OGRGeometry** \*) const
- *Compute intersection.*
- virtual **OGRGeometry** \* **Union** (const **OGRGeometry** \*) const
- *Compute union.*
- virtual **OGRGeometry** \* **UnionCascaded** () const
- *Compute union using cascading.*
- virtual **OGRGeometry** \* **Difference** (const **OGRGeometry** \*) const
- *Compute difference.*
- virtual **OGRGeometry** \* **SymDifference** (const **OGRGeometry** \*) const
- *Compute symmetric difference.*
- virtual OGRErr **Centroid** (**OGRPoint** \*poPoint) const
- *Compute the geometry centroid.*
- virtual **OGRGeometry** \* **Simplify** (double dTolerance) const
- *Simplify the geometry.*
- **OGRGeometry** \* **SimplifyPreserveTopology** (double dTolerance) const
- *Simplify the geometry while preserving topology.*
- virtual **OGRGeometry** \* **Polygonize** () const
- *Polygonizes a set of sparse edges.*
- virtual **OGRGeometry** \* **SymmetricDifference** (const **OGRGeometry** \*) const CPL\_WARN\_DEPRECATED("Non standard method. Use **SymDifference**() instead")
- *Compute symmetric difference (deprecated)*
- virtual **OGRGeometry** \* **getBoundary** () const CPL\_WARN\_DEPRECATED("Non standard method. Use **Boundary**() instead")
- *Compute boundary (deprecated)*
- virtual void **swapXY** ()
- *Swap x and y coordinates.*

### 12.56.1 Detailed Description

Abstract base class for all geometry classes.

Some spatial analysis methods require that OGR is built on the GEOS library to work properly. The precise meaning of methods that describe spatial relationships between geometries is described in the SFCOM, or other simple features interface specifications, like "OpenGIS® Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture" (OGC 06-103r3)



## 12.56.2 Member Function Documentation

### 12.56.2.1 void OGRGeometry::assignSpatialReference ( OGRSpatialReference \* *poSR* )

Assign spatial reference to this object.

Any existing spatial reference is replaced, but under no circumstances does this result in the object being re-projected. It is just changing the interpretation of the existing geometry. Note that assigning a spatial reference increments the reference count on the **OGRSpatialReference** (p. ??), but does not copy it.

This is similar to the SFCOM IGeometry::put\_SpatialReference() method.

This method is the same as the C function **OGR\_G\_AssignSpatialReference()** (p. ??).

#### Parameters

<i>poSR</i>	new spatial reference system to apply.
-------------	--

References OGRSpatialReference::Reference(), and OGRSpatialReference::Release().

Referenced by Boundary(), Buffer(), Centroid(), OGRPoint::clone(), OGRLineString::clone(), OGRLinearRing::clone(), OGRPolygon::clone(), OGRGeometryCollection::clone(), OGRMultiPolygon::clone(), OGRMultiPoint::clone(), OGRMultiLineString::clone(), ConvexHull(), OGRGeometryFactory::createFromWkb(), OGRGeometryFactory::createFromWkt(), Difference(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPoint(), OGRGeometryFactory::forceToMultiPolygon(), OGRGeometryFactory::forceToPolygon(), Intersection(), OGR\_G\_PointOnSurface(), Polygonize(), Simplify(), SimplifyPreserveTopology(), SymDifference(), OGRPoint::transform(), OGRLineString::transform(), OGRPolygon::transform(), OGRGeometryCollection::transform(), Union(), and UnionCascaded().

### 12.56.2.2 OGRGeometry \* OGRGeometry::Boundary ( ) const [virtual]

Compute boundary.

A new geometry object is created and returned containing the boundary of the geometry on which the method is invoked.

This method is the same as the C function **OGR\_G\_Boundary()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

#### Returns

a newly allocated geometry now owned by the caller, or NULL on failure.

#### Since

OGR 1.8.0

References assignSpatialReference(), CPLError(), and getSpatialReference().

Referenced by getBoundary().

### 12.56.2.3 OGRGeometry \* OGRGeometry::Buffer ( double *dfDist*, int *nQuadSegs* = 30 ) const [virtual]

Compute buffer of geometry.

Builds a new geometry containing the buffer region around the geometry on which it is invoked. The buffer is a polygon containing the region within the buffer distance of the original geometry.

Some buffer sections are properly described as curves, but are converted to approximate polygons. The *nQuadSegs* parameter can be used to control how many segments should be used to define a 90 degree curve - a quadrant of a circle. A value of 30 is a reasonable default. Large values result in large numbers of vertices in the resulting buffer geometry while small numbers reduce the accuracy of the result.

This method is the same as the C function **OGR\_G\_Buffer()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a **CPLE\_NotSupported** error.

#### Parameters

<i>dfDist</i>	the buffer distance to be applied.
<i>nQuadSegs</i>	the number of segments used to approximate a 90 degree (quadrant) of curvature.

#### Returns

the newly created geometry, or NULL if an error occurs.

References **assignSpatialReference()**, **CPL\_Error()**, and **getSpatialReference()**.

#### 12.56.2.4 `int OGRGeometry::Centroid ( OGRPoint * poPoint ) const [virtual]`

Compute the geometry centroid.

The centroid location is applied to the passed in **OGRPoint** (p. ??) object. The centroid is not necessarily within the geometry.

This method relates to the **SFCOM ISurface::get\_Centroid()** method however the current implementation based on GEOS can operate on other geometry types such as multipoint, linestring, geometrycollection such as multipolygons. OGC SF SQL 1.1 defines the operation for surfaces (polygons). SQL/MM-Part 3 defines the operation for surfaces and multisurfaces (multipolygons).

This function is the same as the C function **OGR\_G\_Centroid()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a **CPLE\_NotSupported** error.

#### Returns

**OGRERR\_NONE** on success or **OGRERR\_FAILURE** on error.

#### Since

OGR 1.8.0 as a **OGRGeometry** (p. ??) method (previously was restricted to **OGRPolygon** (p. ??))

References **assignSpatialReference()**, **CPL\_Error()**, **OGRPoint::empty()**, **getGeometryType()**, **getSpatialReference()**, **OGRPoint::getX()**, **OGRPoint::getY()**, **OGRPoint::IsEmpty()**, **OGRPoint::setX()**, **OGRPoint::setY()**, and **wkbPoint**.

Referenced by **OGR\_G\_Centroid()**.

#### 12.56.2.5 `OGRGeometry * OGRGeometry::clone ( ) const [pure virtual]`

Make a copy of this object.

This method relates to the **SFCOM IGeometry::clone()** method.

This method is the same as the C function **OGR\_G\_Clone()** (p. ??).

#### Returns

a new object instance with the same geometry, and spatial reference system as the original.

Implemented in **OGRMultiLineString** (p. ??), **OGRMultiPoint** (p. ??), **OGRMultiPolygon** (p. ??), **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), **OGRLinearRing** (p. ??), **OGRLineString** (p. ??), and **OGRPoint** (p. ??).

Referenced by **OGRGeometryCollection::addGeometry()**, **OGRLayer::Clip()**, **OGRWarpedLayer::CreateFeature()**, **OGRLayer::Erase()**, **OGRLayer::Identity()**, **OGRWarpedLayer::SetFeature()**, **OGRFeature::SetGeometry()**, **OGRLayer::SymDifference()**, **OGRLayer::Union()**, and **OGRLayer::Update()**.

**12.56.2.6** void OGRGeometry::closeRings ( ) [virtual]

Force rings to be closed.

If this geometry, or any contained geometries has polygon rings that are not closed, they will be closed by adding the starting point at the end.

Reimplemented in **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), and **OGRLinearRing** (p. ??).

**12.56.2.7** OGRBoolean OGRGeometry::Contains ( const OGRGeometry \* *poOtherGeom* ) const [virtual]

Test for containment.

Tests if actual geometry object contains the passed geometry.

This method is the same as the C function **OGR\_G\_Contains()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

**Parameters**

<i>poOtherGeom</i>	the geometry to compare to this geometry.
--------------------	---

**Returns**

TRUE if *poOtherGeom* contains this geometry, otherwise FALSE.

References CPLError().

**12.56.2.8** OGRGeometry \* OGRGeometry::ConvexHull ( ) const [virtual]

Compute convex hull.

A new geometry object is created and returned containing the convex hull of the geometry on which the method is invoked.

This method is the same as the C function **OGR\_G\_ConvexHull()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

**Returns**

a newly allocated geometry now owned by the caller, or NULL on failure.

References assignSpatialReference(), CPLError(), and getSpatialReference().

**12.56.2.9** OGRBoolean OGRGeometry::Crosses ( const OGRGeometry \* *poOtherGeom* ) const [virtual]

Test for crossing.

Tests if this geometry and the other passed into the method are crossing.

This method is the same as the C function **OGR\_G\_Crosses()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>poOtherGeom</i>	the geometry to compare to this geometry.
--------------------	---

## Returns

TRUE if they are crossing, otherwise FALSE.

References CPLError().

**12.56.2.10 OGRGeometry \* OGRGeometry::Difference ( const OGRGeometry \* *poOtherGeom* ) const** [virtual]

Compute difference.

Generates a new geometry which is the region of this geometry with the region of the second geometry removed.

This method is the same as the C function **OGR\_G\_Difference()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>poOtherGeom</i>	the other geometry removed from "this" geometry.
--------------------	--

## Returns

a new geometry representing the difference or NULL if the difference is empty or an error occurs.

References assignSpatialReference(), CPLError(), getSpatialReference(), and OGRSpatialReference::IsSame().

Referenced by OGRLayer::Erase(), OGRLayer::Identity(), OGRLayer::SymDifference(), OGRLayer::Union(), and OGRLayer::Update().

**12.56.2.11 OGRBoolean OGRGeometry::Disjoint ( const OGRGeometry \* *poOtherGeom* ) const** [virtual]

Test for disjointness.

Tests if this geometry and the other passed into the method are disjoint.

This method is the same as the C function **OGR\_G\_Disjoint()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>poOtherGeom</i>	the geometry to compare to this geometry.
--------------------	---

## Returns

TRUE if they are disjoint, otherwise FALSE.

References CPLError().

**12.56.2.12 double OGRGeometry::Distance ( const OGRGeometry \* *poOtherGeom* ) const** [virtual]

Compute distance between two geometries.

Returns the shortest distance between the two geometries.

This method is the same as the C function **OGR\_G\_Distance()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>poOtherGeom</i>	the other geometry to compare against.
--------------------	--

## Returns

the distance between the geometries or -1 if an error occurs.

References CPLDebug(), and CPLError().

**12.56.2.13** `void OGRGeometry::dumpReadable ( FILE * fp, const char * pszPrefix = NULL, char ** papszOptions = NULL ) const` [virtual]

Dump geometry in well known text format to indicated output file.

A few options can be defined to change the default dump :

- DISPLAY\_GEOMETRY=NO : to hide the dump of the geometry
- DISPLAY\_GEOMETRY=WKT or YES (default) : dump the geometry as a WKT
- DISPLAY\_GEOMETRY=SUMMARY : to get only a summary of the geometry

This method is the same as the C function **OGR\_G\_DumpReadable()** (p. ??).

## Parameters

<i>fp</i>	the text file to write the geometry to.
<i>pszPrefix</i>	the prefix to put on each line of output.
<i>papszOptions</i>	NULL terminated list of options (may be NULL)

References CSLTestBoolean(), dumpReadable(), exportToWkt(), OGRPolygon::getExteriorRing(), getGeometryName(), OGRGeometryCollection::getGeometryRef(), getGeometryType(), OGRPolygon::getInteriorRing(), OGRGeometryCollection::getNumGeometries(), OGRPolygon::getNumInteriorRings(), OGRLineString::getNumPoints(), wkbGeometryCollection, wkbGeometryCollection25D, wkbLinearRing, wkbLineString, wkbLineString25D, wkbMultiLineString, wkbMultiLineString25D, wkbMultiPoint, wkbMultiPoint25D, wkbMultiPolygon, wkbMultiPolygon25D, wkbNone, wkbPoint, wkbPoint25D, wkbPolygon, wkbPolygon25D, and wkbUnknown.

Referenced by dumpReadable(), and OGRFeature::DumpReadable().

**12.56.2.14** `void OGRGeometry::empty ( )` [pure virtual]

Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.

This method relates to the SFCOM IGeometry::Empty() method.

This method is the same as the C function **OGR\_G\_Empty()** (p. ??).

Implemented in **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), **OGRLineString** (p. ??), and **OGRPoint** (p. ??).

**12.56.2.15** `int OGRGeometry::Equals ( OGRGeometry * poOtherGeom ) const` [pure virtual]

Returns TRUE if two geometries are equivalent.

This method is the same as the C function **OGR\_G\_Equals()** (p. ??).

**Returns**

TRUE if equivalent or FALSE otherwise.

Implemented in **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), **OGRLineString** (p. ??), and **OGRPoint** (p. ??).

Referenced by OGRFeature::Equal(), and OGRGeometryCollection::Equals().

**12.56.2.16** `char * OGRGeometry::exportToGML ( const char *const * papszOptions = NULL ) const` [virtual]

Convert a geometry into GML format.

The GML geometry is expressed directly in terms of GML basic data types assuming the this is available in the gml namespace. The returned string should be freed with CPLFree() when no longer required.

The supported options in OGR 1.8.0 are :

- **FORMAT=GML3**. Otherwise it will default to GML 2.1.2 output.
- **GML3\_LINESTRING\_ELEMENT=curve**. (Only valid for **FORMAT=GML3**) To use gml:Curve element for linestrings. Otherwise gml:LineString will be used .
- **GML3\_LONGSRs=YES/NO**. (Only valid for **FORMAT=GML3**) Default to YES. If YES, SRS with EPSG authority will be written with the "urn:ogc:def:crs:EPSG::" prefix. In the case, if the SRS is a geographic SRS without explicit AXIS order, but that the same SRS authority code imported with ImportFromEPSGA() should be treated as lat/long, then the function will take care of coordinate order swapping. If set to NO, SRS with EPSG authority will be written with the "EPSG:" prefix, even if they are in lat/long order.

This method is the same as the C function **OGR\_G\_ExportToGMLEx()** (p. ??).

**Parameters**

<i>papszOptions</i>	NULL-terminated list of options.
---------------------	----------------------------------

**Returns**

A GML fragment or NULL in case of error.

References OGR\_G\_ExportToGMLEx().

**12.56.2.17** `char * OGRGeometry::exportToJson ( ) const` [virtual]

Convert a geometry into GeoJSON format.

The returned string should be freed with CPLFree() when no longer required.

This method is the same as the C function **OGR\_G\_ExportToJson()** (p. ??).

**Returns**

A GeoJSON fragment or NULL in case of error.

References CPLError(), and OGR\_G\_ExportToJson().

**12.56.2.18** `char * OGRGeometry::exportToKML ( ) const` [virtual]

Convert a geometry into KML format.

The returned string should be freed with CPLFree() when no longer required.

This method is the same as the C function **OGR\_G\_ExportToKML()** (p. ??).

**Returns**

A KML fragment or NULL in case of error.

References CPLError(), and OGR\_G\_ExportToKML().

**12.56.2.19** `OGRErr OGRGeometry::exportToWkb ( OGRwkbByteOrder eByteOrder, unsigned char * pabyData ) const`  
[pure virtual]

Convert a geometry into well known binary format.

This method relates to the SFCOM IWks::ExportToWKB() method.

This method is the same as the C function **OGR\_G\_ExportToWkb()** (p. ??).

**Parameters**

<i>eByteOrder</i>	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
<i>pabyData</i>	a buffer into which the binary representation is written. This buffer must be at least <b>OGRGeometry::WkbSize()</b> (p. ??) byte in size.

**Returns**

Currently OGRERR\_NONE is always returned.

Implemented in **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), **OGRLinearRing** (p. ??), **OGRLineString** (p. ??), and **OGRPoint** (p. ??).

Referenced by OGRGeometryCollection::exportToWkb().

**12.56.2.20** `OGRErr OGRGeometry::exportToWkt ( char ** ppszDstText ) const` [pure virtual]

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR\_G\_ExportToWkt()** (p. ??).

**Parameters**

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with OGRFree().
--------------------	--

**Returns**

Currently OGRERR\_NONE is always returned.

Implemented in **OGRMultiLineString** (p. ??), **OGRMultiPoint** (p. ??), **OGRMultiPolygon** (p. ??), **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), **OGRLineString** (p. ??), and **OGRPoint** (p. ??).

Referenced by dumpReadable(), OGRGeometryCollection::exportToWkt(), OGRMultiPolygon::exportToWkt(), OGRMultiLineString::exportToWkt(), and OGRFeature::GetFieldAsString().

**12.56.2.21** `void OGRGeometry::flattenTo2D ( )` [pure virtual]

Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.

This method is the same as the C function **OGR\_G\_FlattenTo2D()** (p. ??).

Implemented in **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), **OGRLineString** (p. ??), and **OGRPoint** (p. ??).

**12.56.2.22** `OGRGeometry * OGRGeometry::getBoundary ( ) const` [virtual]

Compute boundary (deprecated)

### Deprecated

See Also

**Boundary()** (p. ??)

References **Boundary()**.

**12.56.2.23** `int OGRGeometry::getCoordinateDimension ( ) const` [virtual]

Get the dimension of the coordinates in this object.

This method corresponds to the SFCOM IGeometry::GetDimension() method.

This method is the same as the C function **OGR\_G\_GetCoordinateDimension()** (p. ??).

### Returns

in practice this will return 2 or 3. It can also return 0 in the case of an empty point.

Referenced by **OGRGeometryCollection::addGeometryDirectly()**, **OGRLineString::addPoint()**, **OGRPolygon::addRing()**, **OGRPolygon::addRingDirectly()**, **OGRLineString::clone()**, **OGRLineString::exportToWkb()**, **OGRPolygon::exportToWkb()**, **OGRLineString::exportToWkt()**, **OGRPolygon::exportToWkt()**, **OGRMultiPoint::exportToWkt()**, **OGRLineString::getGeometryType()**, **OGRPolygon::getGeometryType()**, **OGRGeometryCollection::getGeometryType()**, **OGRMultiPolygon::getGeometryType()**, **OGRMultiPoint::getGeometryType()**, **OGRMultiLineString::getGeometryType()**, **OGRLineString::getPoint()**, **OGRLineString::segmentize()**, **OGRLineString::setNumPoints()**, **OGRLineString::setPoint()**, **OGRLineString::setPoints()**, **OGRLineString::Value()**, **OGRLineString::WkbSize()**, and **OGRPolygon::WkbSize()**.

**12.56.2.24** `int OGRGeometry::getDimension ( ) const` [pure virtual]

Get the dimension of this object.

This method corresponds to the SFCOM IGeometry::GetDimension() method. It indicates the dimension of the object, but does not indicate the dimension of the underlying space (as indicated by **OGRGeometry::getCoordinateDimension()** (p. ??)).

This method is the same as the C function **OGR\_G\_GetDimension()** (p. ??).

### Returns

0 for points, 1 for lines and 2 for surfaces.

Implemented in **OGRMultiLineString** (p. ??), **OGRMultiPoint** (p. ??), **OGRMultiPolygon** (p. ??), **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), **OGRLineString** (p. ??), and **OGRPoint** (p. ??).

Referenced by **OGRGeometryCollection::getDimension()**, **OGRLayer::Identity()**, **OGRLayer::Intersection()**, and **OGRLayer::Union()**.

**12.56.2.25** `void OGRGeometry::getEnvelope ( OGREnvelope * psEnvelope ) const` [pure virtual]

Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR\_G\_GetEnvelope()** (p. ??).



## Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

Implemented in **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), **OGRLineString** (p. ??), and **OGRPoint** (p. ??).

Referenced by `OGRGeometryCollection::getEnvelope()`, `OGRLayer::GetExtent()`, `OGRLayer::Intersection()`, `Intersects()`, `OGRGeometryFactory::organizePolygons()`, and `OGRWarpedLayer::SetSpatialFilter()`.

**12.56.2.26** `void OGRGeometry::getEnvelope ( OGREnvelope3D * psEnvelope ) const` [pure virtual]

Computes and returns the bounding envelope (3D) for this geometry in the passed *psEnvelope* structure.

This method is the same as the C function **OGR\_G\_GetEnvelope3D()** (p. ??).

## Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

## Since

OGR 1.9.0

Implemented in **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), **OGRLineString** (p. ??), and **OGRPoint** (p. ??).

**12.56.2.27** `const char * OGRGeometry::getGeometryName ( ) const` [pure virtual]

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR\_G\_GetGeometryName()** (p. ??).

## Returns

name used for this geometry type in well known text format. The returned pointer is to a static internal string and should not be modified or freed.

Implemented in **OGRMultiLineString** (p. ??), **OGRMultiPoint** (p. ??), **OGRMultiPolygon** (p. ??), **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), **OGRLinearRing** (p. ??), **OGRLineString** (p. ??), and **OGRPoint** (p. ??).

Referenced by `dumpReadable()`, and `OGRFeature::GetFieldAsString()`.

**12.56.2.28** `OGRwkbGeometryType OGRGeometry::getGeometryType ( ) const` [pure virtual]

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the `wkbFlatten()` macro to the return result.

This method is the same as the C function **OGR\_G\_GetGeometryType()** (p. ??).

## Returns

the geometry type code.

Implemented in **OGRMultiLineString** (p. ??), **OGRMultiPoint** (p. ??), **OGRMultiPolygon** (p. ??), **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), **OGRLineString** (p. ??), and **OGRPoint** (p. ??).

Referenced by `OGRMultiPolygon::addGeometryDirectly()`, `OGRMultiPoint::addGeometryDirectly()`, `OGRMultiLineString::addGeometryDirectly()`, `Centroid()`, `dumpReadable()`, `OGRPoint::Equals()`, `OGRLineString::Equals()`, `OGRPolygon::Equals()`, `OGRGeometryCollection::Equals()`, `OGRGeometryFactory::forceToLineString()`, `OGRGeometryFactory::forceToMultiLineString()`, `OGRGeometryFactory::forceToMultiPoint()`, `OGRGeometryFactory::forceToMultiPolygon()`, `OGRGeometryFactory::forceToPolygon()`, `OGRGeometryCollection::get_Area()`, `OGRGeometryCollection::get_Length()`, `OGR_G_PointOnSurface()`, `OGRBuildPolygonFromEdges()`, and `Polygonize()`.

#### 12.56.2.29 `OGRSpatialReference * OGRGeometry::getSpatialReference ( void ) const` `[inline]`

Returns spatial reference system for object.

This method relates to the SFCOM `IGeometry::get_SpatialReference()` method.

This method is the same as the C function `OGR_G_GetSpatialReference()` (p. ??).

##### Returns

a reference to the spatial reference object. The object may be shared with many geometry objects, and should not be modified.

Referenced by `Boundary()`, `Buffer()`, `Centroid()`, `OGRPoint::clone()`, `OGRLineString::clone()`, `OGRLinearRing::clone()`, `OGRPolygon::clone()`, `OGRGeometryCollection::clone()`, `OGRMultiPolygon::clone()`, `OGRMultiPoint::clone()`, `OGRMultiLineString::clone()`, `ConvexHull()`, `Difference()`, `OGRGeometryFactory::forceToMultiLineString()`, `OGRGeometryFactory::forceToMultiPoint()`, `OGRGeometryFactory::forceToMultiPolygon()`, `OGRGeometryFactory::forceToPolygon()`, `Intersection()`, `OGR_G_PointOnSurface()`, `Polygonize()`, `Simplify()`, `SimplifyPreserveTopology()`, `SymDifference()`, `transformTo()`, `Union()`, and `UnionCascaded()`.

#### 12.56.2.30 `OGRErr OGRGeometry::importFromWkb ( unsigned char * pabyData, int nSize = -1 )` `[pure virtual]`

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM `IWks::ImportFromWKB()` method.

This method is the same as the C function `OGR_G_ImportFromWkb()` (p. ??).

##### Parameters

<i>pabyData</i>	the binary input data.
<i>nSize</i>	the size of pabyData in bytes, or zero if not known.

##### Returns

`OGRERR_NONE` if all goes well, otherwise any of `OGRERR_NOT_ENOUGH_DATA`, `OGRERR_UNSUPPORTED_GEOMETRY_TYPE`, or `OGRERR_CORRUPT_DATA` may be returned.

Implemented in **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), **OGRLinearRing** (p. ??), **OGRLineString** (p. ??), and **OGRPoint** (p. ??).

Referenced by `OGRGeometryFactory::createFromWkb()`.

#### 12.56.2.31 `OGRErr OGRGeometry::importFromWkt ( char ** ppszInput )` `[pure virtual]`

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM `IWks::ImportFromWKT()` method.

This method is the same as the C function `OGR_G_ImportFromWkt()` (p. ??).

## Parameters

<i>ppszInput</i>	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.
------------------	---

## Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

Implemented in **OGRMultiLineString** (p. ??), **OGRMultiPoint** (p. ??), **OGRMultiPolygon** (p. ??), **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), **OGRLineString** (p. ??), and **OGRPoint** (p. ??).

Referenced by OGRGeometryFactory::createFromWkt().

### 12.56.2.32 OGRGeometry \* OGRGeometry::Intersection ( const OGRGeometry \* *poOtherGeom* ) const [virtual]

Compute intersection.

Generates a new geometry which is the region of intersection of the two geometries operated on. The **Intersects()** (p. ??) method can be used to test if two geometries intersect.

This method is the same as the C function **OGR\_G\_Intersection()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>poOtherGeom</i>	the other geometry intersected with "this" geometry.
--------------------	--

## Returns

a new geometry representing the intersection or NULL if there is no intersection or an error occurs.

References assignSpatialReference(), CPLError(), getSpatialReference(), and OGRSpatialReference::IsSame().

Referenced by OGRLayer::Clip(), OGRLayer::Identity(), OGRLayer::Intersection(), and OGRLayer::Union().

### 12.56.2.33 OGRBoolean OGRGeometry::Intersects ( OGRGeometry \* *poOtherGeom* ) const [virtual]

Do these features intersect?

Determines whether two geometries intersect. If GEOS is enabled, then this is done in rigorous fashion otherwise TRUE is returned if the envelopes (bounding boxes) of the two features overlap.

The *poOtherGeom* argument may be safely NULL, but in this case the method will always return TRUE. That is, a NULL geometry is treated as being everywhere.

This method is the same as the C function **OGR\_G\_Intersects()** (p. ??).

## Parameters

<i>poOtherGeom</i>	the other geometry to test against.
--------------------	-------------------------------------

## Returns

TRUE if the geometries intersect, otherwise FALSE.

References getEnvelope().

**12.56.2.34 OGRBoolean OGRGeometry::IsEmpty ( ) const** [pure virtual]

Returns TRUE (non-zero) if the object has no points.

Normally this returns FALSE except between when an object is instantiated and points have been assigned.

This method relates to the SFCOM IGeometry::IsEmpty() method.

**Returns**

TRUE if object is empty, otherwise FALSE.

Implemented in **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), **OGRLineString** (p. ??), and **OGRPoint** (p. ??).

Referenced by OGRLayer::Clip(), OGRLayer::Erase(), OGRLayer::GetExtent(), OGRLayer::Identity(), OGRLayer::Intersection(), OGRLayer::SymDifference(), OGRLayer::Union(), and OGRLayer::Update().

**12.56.2.35 OGRBoolean OGRGeometry::IsRing ( ) const** [virtual]

Test if the geometry is a ring.

This method is the same as the C function **OGR\_G\_IsRing()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always return FALSE.

**Returns**

TRUE if the geometry has no points, otherwise FALSE.

**12.56.2.36 OGRBoolean OGRGeometry::IsSimple ( ) const** [virtual]

Test if the geometry is simple.

This method is the same as the C function **OGR\_G\_IsSimple()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always return FALSE.

**Returns**

TRUE if the geometry has no points, otherwise FALSE.

**12.56.2.37 OGRBoolean OGRGeometry::IsValid ( ) const** [virtual]

Test if the geometry is valid.

This method is the same as the C function **OGR\_G\_IsValid()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always return FALSE.

**Returns**

TRUE if the geometry has no points, otherwise FALSE.

**12.56.2.38** `OGRBoolean OGRGeometry::Overlaps ( const OGRGeometry * poOtherGeom ) const` [virtual]

Test for overlap.

Tests if this geometry and the other passed into the method overlap, that is their intersection has a non-zero area.

This method is the same as the C function **OGR\_G\_Overlaps()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

#### Parameters

<i>poOtherGeom</i>	the geometry to compare to this geometry.
--------------------	---

#### Returns

TRUE if they are overlapping, otherwise FALSE.

References CPLError().

Referenced by OGRGeometryFactory::organizePolygons().

**12.56.2.39** `OGRGeometry * OGRGeometry::Polygonize ( ) const` [virtual]

Polygonizes a set of sparse edges.

A new geometry object is created and returned containing a collection of reassembled Polygons: NULL will be returned if the input collection doesn't corresponds to a MultiLineString, or when reassembling Edges into Polygons is impossible due to topological inconsistencies.

This method is the same as the C function **OGR\_G\_Polygonize()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

#### Returns

a newly allocated geometry now owned by the caller, or NULL on failure.

#### Since

OGR 1.9.0

References assignSpatialReference(), CPLError(), OGRGeometryCollection::getGeometryRef(), getGeometryType(), OGRGeometryCollection::getNumGeometries(), getSpatialReference(), wkbGeometryCollection, wkbLineString, and wkbMultiLineString.

**12.56.2.40** `void OGRGeometry::segmentize ( double dfMaxLength )` [virtual]

Modify the geometry such it has no segment longer then the given distance.

Interpolated points will have Z and M values (if needed) set to 0. Distance computation is performed in 2d only

This function is the same as the C function **OGR\_G\_Segmentize()** (p. ??)

#### Parameters

<i>dfMaxLength</i>	the maximum distance between 2 points after segmentization
--------------------	--

Reimplemented in **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), and **OGRLineString** (p. ??).

#### 12.56.2.41 void OGRGeometry::setCoordinateDimension ( int *nNewDimension* ) [virtual]

Set the coordinate dimension.

This method sets the explicit coordinate dimension. Setting the coordinate dimension of a geometry to 2 should zero out any existing Z values. Setting the dimension of a geometry collection will not necessarily affect the children geometries.

##### Parameters

<i>nNewDimension</i>	New coordinate dimension value, either 2 or 3.
----------------------	--

Reimplemented in **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), **OGRLineString** (p. ??), and **OGRPoint** (p. ??).

Referenced by **OGRPolygon::setCoordinateDimension()**, and **OGRGeometryCollection::setCoordinateDimension()**.

#### 12.56.2.42 OGRGeometry \* OGRGeometry::Simplify ( double *dTolerance* ) const [virtual]

Simplify the geometry.

This function is the same as the C function **OGR\_G\_Simplify()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a **CPL\_NotSupported** error.

##### Parameters

<i>dTolerance</i>	the distance tolerance for the simplification.
-------------------	--

##### Returns

the simplified geometry or NULL if an error occurs.

##### Since

OGR 1.8.0

References **assignSpatialReference()**, **CPL\_Error()**, and **getSpatialReference()**.

#### 12.56.2.43 OGRGeometry \* OGRGeometry::SimplifyPreserveTopology ( double *dTolerance* ) const

Simplify the geometry while preserving topology.

This function is the same as the C function **OGR\_G\_SimplifyPreserveTopology()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a **CPL\_NotSupported** error.

##### Parameters

<i>dTolerance</i>	the distance tolerance for the simplification.
-------------------	--

##### Returns

the simplified geometry or NULL if an error occurs.

Since

OGR 1.9.0

References `assignSpatialReference()`, `CPLError()`, and `getSpatialReference()`.

**12.56.2.44** `void OGRGeometry::swapXY ( ) [virtual]`

Swap x and y coordinates.

Since

OGR 1.8.0

Reimplemented in **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), **OGRLineString** (p. ??), and **OGRPoint** (p. ??).

**12.56.2.45** `OGRGeometry * OGRGeometry::SymDifference ( const OGRGeometry * poOtherGeom ) const [virtual]`

Compute symmetric difference.

Generates a new geometry which is the symmetric difference of this geometry and the second geometry passed into the method.

This method is the same as the C function **OGR\_G\_SymDifference()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a `CPL_NotSupported` error.

Parameters

<i>poOtherGeom</i>	the other geometry.
--------------------	---------------------

Returns

a new geometry representing the symmetric difference or NULL if the difference is empty or an error occurs.

Since

OGR 1.8.0

References `assignSpatialReference()`, `CPLError()`, `getSpatialReference()`, and `OGRSpatialReference::IsSame()`.

Referenced by `SymmetricDifference()`.

**12.56.2.46** `OGRGeometry * OGRGeometry::SymmetricDifference ( const OGRGeometry * poOtherGeom ) const [virtual]`

Compute symmetric difference (deprecated)

**Deprecated**

See Also

**OGRGeometry::SymDifference()** (p. ??)

References `SymDifference()`.

**12.56.2.47** `OGRBoolean OGRGeometry::Touches ( const OGRGeometry * poOtherGeom ) const` `[virtual]`

Test for touching.

Tests if this geometry and the other passed into the method are touching.

This method is the same as the C function **OGR\_G\_Touches()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a `CPLE_NotSupported` error.

#### Parameters

<i>poOtherGeom</i>	the geometry to compare to this geometry.
--------------------	---

#### Returns

TRUE if they are touching, otherwise FALSE.

References `CPLEError()`.

**12.56.2.48** `OGRERR OGRGeometry::transform ( OGRCoordinateTransformation * poCT )` `[pure virtual]`

Apply arbitrary coordinate transformation to geometry.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

Note that this method does not require that the geometry already have a spatial reference system. It will be assumed that they can be treated as having the source spatial reference system of the **OGRCoordinateTransformation** (p. ??) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGRSpatialReference** (p. ??) of the **OGRCoordinateTransformation** (p. ??) will be assigned to the geometry.

This method is the same as the C function **OGR\_G\_Transform()** (p. ??).

#### Parameters

<i>poCT</i>	the transformation to apply.
-------------	------------------------------

#### Returns

`OGRERR_NONE` on success or an error code.

Implemented in **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), **OGRLineString** (p. ??), and **OGRPoint** (p. ??).

Referenced by `OGRWarpedLayer::CreateFeature()`, `OGRWarpedLayer::GetFeature()`, `OGRWarpedLayer::GetNextFeature()`, `OGRWarpedLayer::SetFeature()`, `OGRGeometryCollection::transform()`, and `transformTo()`.

**12.56.2.49** `OGRERR OGRGeometry::transformTo ( OGRSpatialReference * poSR )`

Transform geometry to new spatial reference system.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

This method will only work if the geometry already has an assigned spatial reference system, and if it is transformable to the target coordinate system.

Because this method requires internal creation and initialization of an **OGRCoordinateTransformation** (p. ??) object it is significantly more expensive to use this method to transform many geometries than it is to create the



**OGRCoordinateTransformation** (p. ??) in advance, and call **transform()** (p. ??) with that transformation. This method exists primarily for convenience when only transforming a single geometry.

This method is the same as the C function **OGR\_G\_TransformTo()** (p. ??).

## Parameters

<i>poSR</i>	spatial reference system to transform to.
-------------	---

## Returns

OGRERR\_NONE on success, or an error code.

References `getSpatialReference()`, `OGRCreateCoordinateTransformation()`, and `transform()`.

**12.56.2.50 OGRGeometry \* OGRGeometry::Union ( const OGRGeometry \* *poOtherGeom* ) const** [virtual]

Compute union.

Generates a new geometry which is the region of union of the two geometries operated on.

This method is the same as the C function **OGR\_G\_Union()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>poOtherGeom</i>	the other geometry unioned with "this" geometry.
--------------------	--

## Returns

a new geometry representing the union or NULL if an error occurs.

References `assignSpatialReference()`, `CPLError()`, `getSpatialReference()`, and `OGRSpatialReference::IsSame()`.

Referenced by `OGRLayer::Clip()`, and `OGRLayer::Erase()`.

**12.56.2.51 OGRGeometry \* OGRGeometry::UnionCascaded ( ) const** [virtual]

Compute union using cascading.

This method is the same as the C function **OGR\_G\_UnionCascaded()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

## Returns

a new geometry representing the union or NULL if an error occurs.

## Since

OGR 1.8.0

References `assignSpatialReference()`, `CPLError()`, and `getSpatialReference()`.

**12.56.2.52 OGRBoolean OGRGeometry::Within ( const OGRGeometry \* *poOtherGeom* ) const** [virtual]

Test for containment.

Tests if actual geometry object is within the passed geometry.

This method is the same as the C function **OGR\_G\_Within()** (p. ??).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>poOtherGeom</i>	the geometry to compare to this geometry.
--------------------	---

## Returns

TRUE if poOtherGeom is within this geometry, otherwise FALSE.

References CPLError().

**12.56.2.53** `int OGRGeometry::WkbSize ( ) const` `[pure virtual]`

Returns size of related binary representation.

This method returns the exact number of bytes required to hold the well known binary representation of this geometry object. Its computation may be slightly expensive for complex geometries.

This method relates to the SFCOM IWks::WkbSize() method.

This method is the same as the C function **OGR\_G\_WkbSize()** (p. ??).

## Returns

size of binary representation in bytes.

Implemented in **OGRGeometryCollection** (p. ??), **OGRPolygon** (p. ??), **OGRLinearRing** (p. ??), **OGRLineString** (p. ??), and **OGRPoint** (p. ??).

Referenced by OGRGeometryCollection::exportToWkb(), and OGRGeometryCollection::WkbSize().

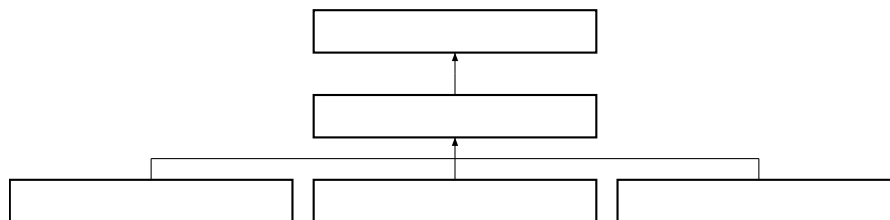
The documentation for this class was generated from the following files:

- **ogr\_geometry.h**
- ogrgeometry.cpp

## 12.57 OGRGeometryCollection Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRGeometryCollection:



### Public Member Functions

- **OGRGeometryCollection ( )**  
*Create an empty geometry collection.*
- virtual const char \* **getGeometryName ( ) const**  
*Fetch WKT name for geometry type.*
- virtual **OGRwkbGeometryType getGeometryType ( ) const**

- Fetch geometry type.*

  - virtual **OGRGeometry \* clone** () const

*Make a copy of this object.*
- virtual void **empty** ()

*Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.*
- virtual OGRErr **transform** (**OGRCoordinateTransformation \*poCT**)

*Apply arbitrary coordinate transformation to geometry.*
- virtual void **flattenTo2D** ()

*Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.*
- virtual OGRBoolean **IsEmpty** () const

*Returns TRUE (non-zero) if the object has no points.*
- virtual void **segmentize** (double dfMaxLength)

*Modify the geometry such it has no segment longer then the given distance.*
- virtual int **WkbSize** () const

*Returns size of related binary representation.*
- virtual OGRErr **importFromWkb** (unsigned char \*, int=-1)

*Assign geometry from well known binary data.*
- virtual OGRErr **exportToWkb** (OGRWkbByteOrder, unsigned char \*) const

*Convert a geometry into well known binary format.*
- virtual OGRErr **importFromWkt** (char \*\*)

*Assign geometry from well known text data.*
- virtual OGRErr **exportToWkt** (char \*\*ppszDstText) const

*Convert a geometry into well known text format.*
- virtual double **get\_Length** () const

*Compute the length of a multcurve.*
- virtual double **get\_Area** () const

*Compute area of geometry collection.*
- virtual int **getDimension** () const

*Get the dimension of this object.*
- virtual void **getEnvelope** (**OGREnvelope \*psEnvelope**) const

*Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.*
- virtual void **getEnvelope** (**OGREnvelope3D \*psEnvelope**) const

*Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.*
- int **getNumGeometries** () const

*Fetch number of geometries in container.*
- **OGRGeometry \* getGeometryRef** (int)

*Fetch geometry from container.*
- virtual OGRBoolean **Equals** (**OGRGeometry \***) const

*Returns TRUE if two geometries are equivalent.*
- virtual void **setCoordinateDimension** (int nDimension)

*Set the coordinate dimension.*
- virtual OGRErr **addGeometry** (const **OGRGeometry \***)

*Add a geometry to the container.*
- virtual OGRErr **addGeometryDirectly** (**OGRGeometry \***)

*Add a geometry directly to the container.*
- virtual OGRErr **removeGeometry** (int iIndex, int bDelete=TRUE)

*Remove a geometry from the container.*
- void **closeRings** ()

*Force rings to be closed.*
- virtual void **swapXY** ()

*Swap x and y coordinates.*

### 12.57.1 Detailed Description

A collection of 1 or more geometry objects.

All geometries must share a common spatial reference system, and Subclasses may impose additional restrictions on the contents.

### 12.57.2 Member Function Documentation

#### 12.57.2.1 OGRErr OGRGeometryCollection::addGeometry ( const OGRGeometry \* *poNewGeom* ) [virtual]

Add a geometry to the container.

Some subclasses of **OGRGeometryCollection** (p. ??) restrict the types of geometry that can be added, and may return an error. The passed geometry is cloned to make an internal copy.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR\_G\_AddGeometry()** (p. ??).

##### Parameters

<i>poNewGeom</i>	geometry to add to the container.
------------------	-----------------------------------

##### Returns

OGRERR\_NONE if successful, or OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE if the geometry type is illegal for the type of geometry container.

References `addGeometryDirectly()`, and `OGRGeometry::clone()`.

Referenced by `clone()`, `OGRMultiPolygon::clone()`, `OGRMultiPoint::clone()`, and `OGRMultiLineString::clone()`.

#### 12.57.2.2 OGRErr OGRGeometryCollection::addGeometryDirectly ( OGRGeometry \* *poNewGeom* ) [virtual]

Add a geometry directly to the container.

Some subclasses of **OGRGeometryCollection** (p. ??) restrict the types of geometry that can be added, and may return an error. Ownership of the passed geometry is taken by the container rather than cloning as **addGeometry()** (p. ??) does.

This method is the same as the C function **OGR\_G\_AddGeometryDirectly()** (p. ??).

There is no SFCOM analog to this method.

##### Parameters

<i>poNewGeom</i>	geometry to add to the container.
------------------	-----------------------------------

##### Returns

OGRERR\_NONE if successful, or OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE if the geometry type is illegal for the type of geometry container.

Reimplemented in **OGRMultiLineString** (p. ??), **OGRMultiPoint** (p. ??), and **OGRMultiPolygon** (p. ??).

References `OGRGeometry::getCoordinateDimension()`.

Referenced by `addGeometry()`, `OGRMultiPolygon::addGeometryDirectly()`, `OGRMultiPoint::addGeometryDirectly()`, and `OGRMultiLineString::addGeometryDirectly()`.

### 12.57.2.3 **OGRGeometry \* OGRGeometryCollection::clone ( ) const** [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR\_G\_Clone()** (p. ??).

#### Returns

a new object instance with the same geometry, and spatial reference system as the original.

Implements **OGRGeometry** (p. ??).

Reimplemented in **OGRMultiLineString** (p. ??), **OGRMultiPoint** (p. ??), and **OGRMultiPolygon** (p. ??).

References **addGeometry()**, **OGRGeometry::assignSpatialReference()**, **OGRGeometry::getSpatialReference()**, and **OGRGeometryCollection()**.

### 12.57.2.4 **void OGRGeometryCollection::closeRings ( )** [virtual]

Force rings to be closed.

If this geometry, or any contained geometries has polygon rings that are not closed, they will be closed by adding the starting point at the end.

Reimplemented from **OGRGeometry** (p. ??).

References **getGeometryType()**, and **wkbPolygon**.

### 12.57.2.5 **void OGRGeometryCollection::empty ( )** [virtual]

Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.

This method relates to the SFCOM IGeometry::Empty() method.

This method is the same as the C function **OGR\_G\_Empty()** (p. ??).

Implements **OGRGeometry** (p. ??).

Referenced by **OGRMultiPolygon::importFromWkt()**, **OGRMultiPoint::importFromWkt()**, and **OGRMultiLineString::importFromWkt()**.

### 12.57.2.6 **OGRBoolean OGRGeometryCollection::Equals ( OGRGeometry \* poOtherGeom ) const** [virtual]

Returns TRUE if two geometries are equivalent.

This method is the same as the C function **OGR\_G\_Equals()** (p. ??).

#### Returns

TRUE if equivalent or FALSE otherwise.

Implements **OGRGeometry** (p. ??).

References **OGRGeometry::Equals()**, **getGeometryRef()**, **OGRGeometry::getGeometryType()**, **getGeometryType()**, and **getNumGeometries()**.

### 12.57.2.7 **OGRErr OGRGeometryCollection::exportToWkb ( OGRwkbByteOrder eByteOrder, unsigned char \* pabyData ) const** [virtual]

Convert a geometry into well known binary format.

This method relates to the SFCOM IWks::ExportToWKB() method.

This method is the same as the C function **OGR\_G\_ExportToWkb()** (p. ??).

#### Parameters

<i>eByteOrder</i>	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
<i>pabyData</i>	a buffer into which the binary representation is written. This buffer must be at least <b>OGRGeometry::WkbSize()</b> (p. ??) byte in size.

#### Returns

Currently OGRERR\_NONE is always returned.

Implements **OGRGeometry** (p. ??).

References OGRGeometry::exportToWkb(), getGeometryType(), and OGRGeometry::WkbSize().

#### 12.57.2.8 OGRErr OGRGeometryCollection::exportToWkt ( char \*\* ppszDstText ) const [virtual]

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR\_G\_ExportToWkt()** (p. ??).

#### Parameters

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with OGRFree().
--------------------	--

#### Returns

Currently OGRERR\_NONE is always returned.

Implements **OGRGeometry** (p. ??).

Reimplemented in **OGRMultiLineString** (p. ??), **OGRMultiPoint** (p. ??), and **OGRMultiPolygon** (p. ??).

References CPLCalloc(), CPLStrdup(), OGRGeometry::exportToWkt(), getGeometryName(), and getNumGeometries().

#### 12.57.2.9 void OGRGeometryCollection::flattenTo2D ( ) [virtual]

Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.

This method is the same as the C function **OGR\_G\_FlattenTo2D()** (p. ??).

Implements **OGRGeometry** (p. ??).

#### 12.57.2.10 double OGRGeometryCollection::get\_Area ( ) const [virtual]

Compute area of geometry collection.

The area is computed as the sum of the areas of all members in this collection.

#### Note

No warning will be issued if a member of the collection does not support the get\_Area method.

**Returns**

computed area.

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

References `getGeometryName()`, `OGRGeometry::getGeometryType()`, `wkbGeometryCollection`, `wkbLinearRing`, `wkbLineString`, `wkbMultiPolygon`, and `wkbPolygon`.

### 12.57.2.11 `double OGRGeometryCollection::get_Length ( ) const [virtual]`

Compute the length of a multicurve.

The length is computed as the sum of the length of all members in this collection.

**Note**

No warning will be issued if a member of the collection does not support the `get_Length` method.

**Returns**

computed area.

References `OGRGeometry::getGeometryType()`, `wkbGeometryCollection`, `wkbLinearRing`, and `wkbLineString`.

### 12.57.2.12 `int OGRGeometryCollection::getDimension ( ) const [virtual]`

Get the dimension of this object.

This method corresponds to the SFCOM `IGeometry::GetDimension()` method. It indicates the dimension of the object, but does not indicate the dimension of the underlying space (as indicated by **`OGRGeometry::getCoordinateDimension()`** (p. ??)).

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

**Returns**

0 for points, 1 for lines and 2 for surfaces.

Implements **`OGRGeometry`** (p. ??).

Reimplemented in **`OGRMultiLineString`** (p. ??), **`OGRMultiPoint`** (p. ??), and **`OGRMultiPolygon`** (p. ??).

References `OGRGeometry::getDimension()`.

### 12.57.2.13 `void OGRGeometryCollection::getEnvelope ( OGREnvelope * psEnvelope ) const [virtual]`

Computes and returns the bounding envelope for this geometry in the passed `psEnvelope` structure.

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

**Parameters**

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

Implements **`OGRGeometry`** (p. ??).

References `OGRGeometry::getEnvelope()`, and `IsEmpty()`.

### 12.57.2.14 `void OGRGeometryCollection::getEnvelope ( OGREnvelope3D * psEnvelope ) const [virtual]`

Computes and returns the bounding envelope (3D) for this geometry in the passed `psEnvelope` structure.

This method is the same as the C function **`OGR_G_GetEnvelope3D()`** (p. ??).



## Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

## Since

OGR 1.9.0

Implements **OGRGeometry** (p. ??).

References OGRGeometry::getEnvelope(), and IsEmpty().

### 12.57.2.15 `const char * OGRGeometryCollection::getGeometryName ( ) const` [virtual]

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR\_G\_GetGeometryName()** (p. ??).

## Returns

name used for this geometry type in well known text format. The returned pointer is to a static internal string and should not be modified or freed.

Implements **OGRGeometry** (p. ??).

Reimplemented in **OGRMultiLineString** (p. ??), **OGRMultiPoint** (p. ??), and **OGRMultiPolygon** (p. ??).

Referenced by exportToWkt(), and get\_Area().

### 12.57.2.16 `OGRGeometry * OGRGeometryCollection::getGeometryRef ( int i )`

Fetch geometry from container.

This method returns a pointer to an geometry within the container. The returned geometry remains owned by the container, and should not be modified. The pointer is only valid untill the next change to the geometry container. Use IGeometry::clone() to make a copy.

This method relates to the SFCOM IGeometryCollection::get\_Geometry() method.

## Parameters

<i>i</i>	the index of the geometry to fetch, between 0 and <b>getNumGeometries()</b> (p. ??) - 1.
----------	--

## Returns

pointer to requested geometry.

Referenced by OGRMultiPolygon::clone(), OGRMultiPoint::clone(), OGRMultiLineString::clone(), OGRGeometry::dumpReadable(), Equals(), OGRMultiPolygon::exportToWkt(), OGRMultiPoint::exportToWkt(), OGRMultiLineString::exportToWkt(), OGRGeometryFactory::forceToLineString(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPoint(), OGRGeometryFactory::forceToMultiPolygon(), OGRGeometryFactory::forceToPolygon(), OGRMultiPolygon::get\_Area(), OGRBuildPolygonFromEdges(), and OGRGeometry::Polygonize().

### 12.57.2.17 `OGRwkbGeometryType OGRGeometryCollection::getGeometryType ( ) const` [virtual]

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the wkbFlatten() macro to the return result.

This method is the same as the C function **OGR\_G\_GetGeometryType()** (p. ??).

**Returns**

the geometry type code.

Implements **OGRGeometry** (p. ??).

Reimplemented in **OGRMultiLineString** (p. ??), **OGRMultiPoint** (p. ??), and **OGRMultiPolygon** (p. ??).

References OGRGeometry::getCoordinateDimension(), wkbGeometryCollection, and wkbGeometryCollection25D.

Referenced by closeRings(), Equals(), and exportToWkb().

**12.57.2.18 int OGRGeometryCollection::getNumGeometries ( ) const**

Fetch number of geometries in container.

This method relates to the SFCOM IGeometryCollect::get\_NumGeometries() method.

**Returns**

count of children geometries. May be zero.

Referenced by OGRMultiPolygon::clone(), OGRMultiPoint::clone(), OGRMultiLineString::clone(), OGRGeometry::dumpReadable(), Equals(), exportToWkt(), OGRMultiPolygon::exportToWkt(), OGRMultiPoint::exportToWkt(), OGRMultiLineString::exportToWkt(), OGRGeometryFactory::forceToLineString(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPoint(), OGRGeometryFactory::forceToMultiPolygon(), OGRGeometryFactory::forceToPolygon(), OGRMultiPolygon::get\_Area(), OGRBuildPolygonFromEdges(), and OGRGeometry::Polygonize().

**12.57.2.19 OGRErr OGRGeometryCollection::importFromWkb ( unsigned char \* *pabyData*, int *nSize* = -1 ) [virtual]**

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKB() method.

This method is the same as the C function **OGR\_G\_ImportFromWkb()** (p. ??).

**Parameters**

<i>pabyData</i>	the binary input data.
<i>nSize</i>	the size of pabyData in bytes, or zero if not known.

**Returns**

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

Implements **OGRGeometry** (p. ??).

**12.57.2.20 OGRErr OGRGeometryCollection::importFromWkt ( char \*\* *ppszInput* ) [virtual]**

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR\_G\_ImportFromWkt()** (p. ??).

## Parameters

<i>ppszInput</i>	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.
------------------	---

## Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

Implements **OGRGeometry** (p. ??).

Reimplemented in **OGRMultiLineString** (p. ??), **OGRMultiPoint** (p. ??), and **OGRMultiPolygon** (p. ??).

#### 12.57.2.21 OGRBoolean OGRGeometryCollection::IsEmpty ( ) const [virtual]

Returns TRUE (non-zero) if the object has no points.

Normally this returns FALSE except between when an object is instantiated and points have been assigned.

This method relates to the SFCOM IGeometry::IsEmpty() method.

## Returns

TRUE if object is empty, otherwise FALSE.

Implements **OGRGeometry** (p. ??).

Referenced by OGRMultiPoint::exportToWkt(), and getEnvelope().

#### 12.57.2.22 OGRErr OGRGeometryCollection::removeGeometry ( int *iGeom*, int *bDelete* = TRUE ) [virtual]

Remove a geometry from the container.

Removing a geometry will cause the geometry count to drop by one, and all "higher" geometries will shuffle down one in index.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR\_G\_RemoveGeometry()** (p. ??).

## Parameters

<i>iGeom</i>	the index of the geometry to delete. A value of -1 is a special flag meaning that all geometries should be removed.
<i>bDelete</i>	if TRUE the geometry will be deallocated, otherwise it will not. The default is TRUE as the container is considered to own the geometries in it.

## Returns

OGRERR\_NONE if successful, or OGRERR\_FAILURE if the index is out of range.

Referenced by OGRGeometryFactory::forceToLineString(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToMultiPoint(), and OGRGeometryFactory::forceToMultiPolygon().

#### 12.57.2.23 void OGRGeometryCollection::segmentize ( double *dfMaxLength* ) [virtual]

Modify the geometry such it has no segment longer then the given distance.

Interpolated points will have Z and M values (if needed) set to 0. Distance computation is performed in 2d only

This function is the same as the C function **OGR\_G\_Segmentize()** (p. ??)

## Parameters

<i>dfMaxLength</i>	the maximum distance between 2 points after segmentization
--------------------	--

Reimplemented from **OGRGeometry** (p. ??).

**12.57.2.24** void OGRGeometryCollection::setCoordinateDimension ( int *nNewDimension* ) [virtual]

Set the coordinate dimension.

This method sets the explicit coordinate dimension. Setting the coordinate dimension of a geometry to 2 should zero out any existing Z values. Setting the dimension of a geometry collection will not necessarily affect the children geometries.

## Parameters

<i>nNewDimension</i>	New coordinate dimension value, either 2 or 3.
----------------------	--

Reimplemented from **OGRGeometry** (p. ??).

References OGRGeometry::setCoordinateDimension().

**12.57.2.25** void OGRGeometryCollection::swapXY ( ) [virtual]

Swap x and y coordinates.

## Since

OGR 1.8.0

Reimplemented from **OGRGeometry** (p. ??).

**12.57.2.26** OGRErr OGRGeometryCollection::transform ( OGRCoordinateTransformation \* *poCT* ) [virtual]

Apply arbitrary coordinate transformation to geometry.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

Note that this method does not require that the geometry already have a spatial reference system. It will be assumed that they can be treated as having the source spatial reference system of the **OGRCoordinateTransformation** (p. ??) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGRSpatialReference** (p. ??) of the **OGRCoordinateTransformation** (p. ??) will be assigned to the geometry.

This method is the same as the C function **OGR\_G\_Transform()** (p. ??).

## Parameters

<i>poCT</i>	the transformation to apply.
-------------	------------------------------

## Returns

OGRERR\_NONE on success or an error code.

Implements **OGRGeometry** (p. ??).

References OGRGeometry::assignSpatialReference(), CPLDebug(), OGRCoordinateTransformation::GetTargetC-S(), and OGRGeometry::transform().

12.57.2.27 `int OGRGeometryCollection::WkbSize ( ) const [virtual]`

Returns size of related binary representation.

This method returns the exact number of bytes required to hold the well known binary representation of this geometry object. Its computation may be slightly expensive for complex geometries.

This method relates to the SFCOM IWks::WkbSize() method.

This method is the same as the C function **OGR\_G\_WkbSize()** (p. ??).

#### Returns

size of binary representation in bytes.

Implements **OGRGeometry** (p. ??).

References **OGRGeometry::WkbSize()**.

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

- **ogr\_geometry.h**
- **ogrgeometrycollection.cpp**

## 12.58 OGRGeometryFactory Class Reference

```
#include <ogr_geometry.h>
```

### Static Public Member Functions

- static **OGRERR** **createFromWkb** (unsigned char \*, **OGRSpatialReference** \*, **OGRGeometry** \*\*, int=-1)  
*Create a geometry object of the appropriate type from it's well known binary representation.*
- static **OGRERR** **createFromWkt** (char \*\*, **OGRSpatialReference** \*, **OGRGeometry** \*\*)  
*Create a geometry object of the appropriate type from it's well known text representation.*
- static **OGRERR** **createFromFgf** (unsigned char \*, **OGRSpatialReference** \*, **OGRGeometry** \*\*, int=-1, int \*=NULL)  
*Create a geometry object of the appropriate type from it's FGF (FDO Geometry Format) binary representation.*
- static **OGRGeometry** \* **createFromGML** (const char \*)  
*Create geometry from GML.*
- static void **destroyGeometry** (**OGRGeometry** \*)  
*Destroy geometry object.*
- static **OGRGeometry** \* **createGeometry** (**OGRwkbGeometryType**)  
*Create an empty geometry of desired type.*
- static **OGRGeometry** \* **forceToPolygon** (**OGRGeometry** \*)  
*Convert to polygon.*
- static **OGRGeometry** \* **forceToLineString** (**OGRGeometry** \*, bool bOnlyInOrder=true)  
*Convert to line string.*
- static **OGRGeometry** \* **forceToMultiPolygon** (**OGRGeometry** \*)  
*Convert to multipolygon.*
- static **OGRGeometry** \* **forceToMultiPoint** (**OGRGeometry** \*)  
*Convert to multipoint.*
- static **OGRGeometry** \* **forceToMultiLineString** (**OGRGeometry** \*)  
*Convert to multilinestring.*
- static **OGRGeometry** \* **organizePolygons** (**OGRGeometry** \*\*papoPolygons, int nPolygonCount, int \*pResultValidGeometry, const char \*\*papszOptions=NULL)

*Organize polygons based on geometries.*

- static int **haveGEOS** ()

*Test if GEOS enabled.*

- static **OGRGeometry \* approximateArcAngles** (double dfX, double dfY, double dfZ, double dfPrimaryRadius, double dfSecondaryAxis, double dfRotation, double dfStartAngle, double dfEndAngle, double dfMaxAngleStepSizeDegrees)

### 12.58.1 Detailed Description

Create geometry objects from well known text/binary.

### 12.58.2 Member Function Documentation

12.58.2.1 **OGRGeometry \* OGRGeometryFactory::approximateArcAngles** ( double dfCenterX, double dfCenterY, double dfZ, double dfPrimaryRadius, double dfSecondaryRadius, double dfRotation, double dfStartAngle, double dfEndAngle, double dfMaxAngleStepSizeDegrees ) [static]

Stroke arc to linestring.

Stroke an arc of a circle to a linestring based on a center point, radius, start angle and end angle, all angles in degrees.

If the dfMaxAngleStepSizeDegrees is zero, then a default value will be used. This is currently 4 degrees unless the user has overridden the value with the OGR\_ARC\_STEPSIZE configuration variable.

See Also

**CPLSetConfigOption()** (p. ??)

#### Parameters

<i>dfCenterX</i>	center X
<i>dfCenterY</i>	center Y
<i>dfZ</i>	center Z
<i>dfPrimaryRadius</i>	X radius of ellipse.
<i>dfSecondaryRadius</i>	Y radius of ellipse.
<i>dfRotation</i>	rotation of the ellipse clockwise.
<i>dfStartAngle</i>	angle to first point on arc (clockwise of X-positive)
<i>dfEndAngle</i>	angle to last point on arc (clockwise of X-positive)
<i>dfMaxAngleStepSizeDegrees</i>	the largest step in degrees along the arc, zero to use the default setting.

#### Returns

**OGRLineString** (p. ??) geometry representing an approximation of the arc.

Since

OGR 1.8.0

References **CPLGetConfigOption()**, and **OGRLineString::setPoint()**.

Referenced by **OGR\_G\_ApproximateArcAngles()**.

**12.58.2.2** `OGRERR OGRGeometryFactory::createFromFgf ( unsigned char * pabyData, OGRSpatialReference * poSR, OGRGeometry ** ppoReturn, int nBytes = -1, int * pnBytesConsumed = NULL ) [static]`

Create a geometry object of the appropriate type from it's FGF (FDO Geometry Format) binary representation.

Also note that this is a static method, and that there is no need to instantiate an **OGRGeometryFactory** (p. ??) object.

The C function `OGR_G_CreateFromFgf()` is the same as this method.

#### Parameters

<i>pabyData</i>	pointer to the input BLOB data.
<i>poSR</i>	pointer to the spatial reference to be assigned to the created geometry object. This may be NULL.
<i>ppoReturn</i>	the newly created geometry object will be assigned to the indicated pointer on return. This will be NULL in case of failure.
<i>nBytes</i>	the number of bytes available in <i>pabyData</i> .
<i>pnBytesConsumed</i>	if not NULL, it will be set to the number of bytes consumed (at most <i>nBytes</i> ).

#### Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

**12.58.2.3** `OGRGeometry * OGRGeometryFactory::createFromGML ( const char * pszData ) [static]`

Create geometry from GML.

This method translates a fragment of GML containing only the geometry portion into a corresponding **OGRGeometry** (p. ??). There are many limitations on the forms of GML geometries supported by this parser, but they are too numerous to list here.

The following GML2 elements are parsed : Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, MultiGeometry.

(OGR >= 1.8.0) The following GML3 elements are parsed : Surface, MultiSurface, PolygonPatch, Triangle, Rectangle, Curve, MultiCurve, LineStringSegment, Arc, Circle, CompositeSurface, OrientableSurface, Solid, Tin, TriangulatedSurface.

Arc and Circle elements are stroked to linestring, by using a 4 degrees step, unless the user has overridden the value with the `OGR_ARC_STEPSIZE` configuration variable.

The C function **OGR\_G\_CreateFromGML()** (p. ??) is the same as this method.

#### Parameters

<i>pszData</i>	The GML fragment for the geometry.
----------------	------------------------------------

#### Returns

a geometry on succes, or NULL on error.

References `OGR_G_CreateFromGML()`.

**12.58.2.4** `OGRERR OGRGeometryFactory::createFromWkb ( unsigned char * pabyData, OGRSpatialReference * poSR, OGRGeometry ** ppoReturn, int nBytes = -1 ) [static]`

Create a geometry object of the appropriate type from it's well known binary representation.

Note that if `nBytes` is passed as zero, no checking can be done on whether the `pabyData` is sufficient. This can result in a crash if the input data is corrupt. This function returns no indication of the number of bytes from the data source actually used to represent the returned geometry object. Use **`OGRGeometry::WkbSize()`** (p. ??) on the returned geometry to establish the number of bytes it required in WKB format.

Also note that this is a static method, and that there is no need to instantiate an **`OGRGeometryFactory`** (p. ??) object.

The C function **`OGR_G_CreateFromWkb()`** (p. ??) is the same as this method.

#### Parameters

<i>pabyData</i>	pointer to the input BLOB data.
<i>poSR</i>	pointer to the spatial reference to be assigned to the created geometry object. This may be NULL.
<i>ppoReturn</i>	the newly created geometry object will be assigned to the indicated pointer on return. This will be NULL in case of failure. If not NULL, *ppoReturn should be freed with <b><code>OGRGeometryFactory::destroyGeometry()</code></b> (p. ??) after use.
<i>nBytes</i>	the number of bytes available in <code>pabyData</code> , or -1 if it isn't known.

#### Returns

`OGRERR_NONE` if all goes well, otherwise any of `OGRERR_NOT_ENOUGH_DATA`, `OGRERR_UNSUPPORTED_GEOMETRY_TYPE`, or `OGRERR_CORRUPT_DATA` may be returned.

References `OGRGeometry::assignSpatialReference()`, `CPLDebug()`, `createGeometry()`, and `OGRGeometry::importFromWkb()`.

Referenced by `OGR_G_CreateFromWkb()`.

**12.58.2.5** `OGRERR OGRGeometryFactory::createFromWkt ( char ** ppszData, OGRSpatialReference * poSR, OGRGeometry ** ppoReturn ) [static]`

Create a geometry object of the appropriate type from it's well known text representation.

The C function **`OGR_G_CreateFromWkt()`** (p. ??) is the same as this method.

#### Parameters

<i>ppszData</i>	input zero terminated string containing well known text representation of the geometry to be created. The pointer is updated to point just beyond that last character consumed.
<i>poSR</i>	pointer to the spatial reference to be assigned to the created geometry object. This may be NULL.
<i>ppoReturn</i>	the newly created geometry object will be assigned to the indicated pointer on return. This will be NULL if the method fails. If not NULL, *ppoReturn should be freed with <b><code>OGRGeometryFactory::destroyGeometry()</code></b> (p. ??) after use.

#### Example:

```
const char* wkt= "POINT(0 0)";

// cast because OGR_G_CreateFromWkt will move the pointer
char* pszWkt = (char*) wkt;
OGRSpatialReferenceH ref = OSRNewSpatialReference(NULL);
OGRGeometryH new_geom;
OGRERR err = OGR_G_CreateFromWkt(&pszWkt, ref, &new_geom);
```



**Returns**

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

References OGRGeometry::assignSpatialReference(), and OGRGeometry::importFromWkt().

Referenced by OGR\_G\_CreateFromWkt().

#### 12.58.2.6 OGRGeometry \* OGRGeometryFactory::createGeometry ( OGRwkbGeometryType eGeometryType ) [static]

Create an empty geometry of desired type.

This is equivalent to allocating the desired geometry with new, but the allocation is guaranteed to take place in the context of the GDAL/OGR heap.

This method is the same as the C function **OGR\_G\_CreateGeometry()** (p. ??).

**Parameters**

<i>eGeometryType</i>	the type code of the geometry class to be instantiated.
----------------------	---

**Returns**

the newly create geometry or NULL on failure. Should be freed with **OGRGeometryFactory::destroyGeometry()** (p. ??) after use.

References wkbGeometryCollection, wkbLinearRing, wkbLineString, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, wkbPoint, and wkbPolygon.

Referenced by createFromWkb(), and OGR\_G\_CreateGeometry().

#### 12.58.2.7 void OGRGeometryFactory::destroyGeometry ( OGRGeometry \* poGeom ) [static]

Destroy geometry object.

Equivalent to invoking delete on a geometry, but it guaranteed to take place within the context of the GDAL/OGR heap.

This method is the same as the C function **OGR\_G\_DestroyGeometry()** (p. ??).

**Parameters**

<i>poGeom</i>	the geometry to deallocate.
---------------	-----------------------------

Referenced by OGR\_G\_DestroyGeometry().

#### 12.58.2.8 OGRGeometry \* OGRGeometryFactory::forceToLineString ( OGRGeometry \* poGeom, bool bOnlyInOrder = true ) [static]

Convert to line string.

Tries to force the provided geometry to be a line string. Currently this just effects a change on multilinestrings. The passed in geometry is consumed and a new one returned (or potentially the same one).

**Parameters**

<i>poGeom</i>	the input geometry - ownership is passed to the method.
---------------	---

**Returns**

new geometry.

References `OGRLineString::addSubLineString()`, `OGRLineString::EndPoint()`, `OGRPoint::Equals()`, `OGRGeometryCollection::getGeometryRef()`, `OGRGeometry::getGeometryType()`, `OGRGeometryCollection::getNumGeometries()`, `OGRLineString::getNumPoints()`, `OGRGeometryCollection::removeGeometry()`, `OGRLineString::reversePoints()`, `OGRLineString::StartPoint()`, `wkbGeometryCollection`, `wkbLineString`, and `wkbMultiLineString`.

Referenced by `OGR_G_ForceToLineString()`.

### 12.58.2.9 `OGRGeometry * OGRGeometryFactory::forceToMultiLineString ( OGRGeometry * poGeom ) [static]`

Convert to multilinestring.

Tries to force the provided geometry to be a multilinestring.

- linestrings are placed in a multilinestring.
- geometry collections will be converted to multilinestring if they only contain linestrings.
- polygons will be changed to a collection of linestrings (one per ring).

The passed in geometry is consumed and a new one returned (or potentially the same one).

**Returns**

new geometry.

References `OGRMultiLineString::addGeometryDirectly()`, `OGRLineString::addSubLineString()`, `OGRGeometry::assignSpatialReference()`, `OGRPolygon::getExteriorRing()`, `OGRGeometryCollection::getGeometryRef()`, `OGRGeometry::getGeometryType()`, `OGRPolygon::getInteriorRing()`, `OGRGeometryCollection::getNumGeometries()`, `OGRPolygon::getNumInteriorRings()`, `OGRLineString::getNumPoints()`, `OGRGeometry::getSpatialReference()`, `OGRGeometryCollection::removeGeometry()`, `wkbGeometryCollection`, `wkbLineString`, `wkbMultiLineString`, `wkbMultiPolygon`, and `wkbPolygon`.

Referenced by `OGR_G_ForceToMultiLineString()`.

### 12.58.2.10 `OGRGeometry * OGRGeometryFactory::forceToMultiPoint ( OGRGeometry * poGeom ) [static]`

Convert to multipoint.

Tries to force the provided geometry to be a multipoint. Currently this just effects a change on points. The passed in geometry is consumed and a new one returned (or potentially the same one).

**Returns**

new geometry.

References `OGRMultiPoint::addGeometryDirectly()`, `OGRGeometry::assignSpatialReference()`, `OGRGeometryCollection::getGeometryRef()`, `OGRGeometry::getGeometryType()`, `OGRGeometryCollection::getNumGeometries()`, `OGRGeometry::getSpatialReference()`, `OGRGeometryCollection::removeGeometry()`, `wkbGeometryCollection`, `wkbMultiPoint`, and `wkbPoint`.

Referenced by `OGR_G_ForceToMultiPoint()`.

**12.58.2.11 OGRGeometry \* OGRGeometryFactory::forceToMultiPolygon ( OGRGeometry \* *poGeom* ) [static]**

Convert to multipolygon.

Tries to force the provided geometry to be a multipolygon. Currently this just effects a change on polygons. The passed in geometry is consumed and a new one returned (or potentially the same one).

**Returns**

new geometry.

References OGRMultiPolygon::addGeometryDirectly(), OGRGeometry::assignSpatialReference(), OGRGeometryCollection::getGeometryRef(), OGRGeometry::getGeometryType(), OGRGeometryCollection::getNumGeometries(), OGRGeometry::getSpatialReference(), OGRGeometryCollection::removeGeometry(), wkbGeometryCollection, wkbMultiPolygon, and wkbPolygon.

Referenced by OGR\_G\_ForceToMultiPolygon().

**12.58.2.12 OGRGeometry \* OGRGeometryFactory::forceToPolygon ( OGRGeometry \* *poGeom* ) [static]**

Convert to polygon.

Tries to force the provided geometry to be a polygon. Currently this just effects a change on multipolygons. The passed in geometry is consumed and a new one returned (or potentially the same one).

**Parameters**

<i>poGeom</i>	the input geometry - ownership is passed to the method.
---------------	---

**Returns**

new geometry.

References OGRPolygon::addRing(), OGRGeometry::assignSpatialReference(), OGRPolygon::getExteriorRing(), OGRGeometryCollection::getGeometryRef(), OGRGeometry::getGeometryType(), OGRPolygon::getInteriorRing(), OGRGeometryCollection::getNumGeometries(), OGRPolygon::getNumInteriorRings(), OGRGeometry::getSpatialReference(), wkbGeometryCollection, wkbMultiPolygon, and wkbPolygon.

Referenced by OGR\_G\_ForceToPolygon().

**12.58.2.13 int OGRGeometryFactory::haveGEOS ( ) [static]**

Test if GEOS enabled.

This static method returns TRUE if GEOS support is built into OGR, otherwise it returns FALSE.

**Returns**

TRUE if available, otherwise FALSE.

Referenced by OGRLayer::Clip(), OGRLayer::Erase(), OGRLayer::Identity(), OGRLayer::Intersection(), organizePolygons(), OGRLayer::SymDifference(), OGRLayer::Union(), and OGRLayer::Update().

**12.58.2.14 OGRGeometry \* OGRGeometryFactory::organizePolygons ( OGRGeometry \*\* *papoPolygons*, int *nPolygonCount*, int \* *pblsValidGeometry*, const char \*\* *papszOptions* = NULL ) [static]**

Organize polygons based on geometries.

Analyse a set of rings (passed as simple polygons), and based on a geometric analysis convert them into a polygon with inner rings, or a MultiPolygon if dealing with more than one polygon.

All the input geometries must be OGRPolygons with only a valid exterior ring (at least 4 points) and no interior rings. The passed in geometries become the responsibility of the method, but the `papoPolygons` "pointer array" remains owned by the caller.

For faster computation, a polygon is considered to be inside another one if a single point of its external ring is included into the other one. (unless 'OGR\_DEBUG\_ORGANIZE\_POLYGONS' configuration option is set to TRUE. In that case, a slower algorithm that tests exact topological relationships is used if GEOS is available.)

In cases where a big number of polygons is passed to this function, the default processing may be really slow. You can skip the processing by adding `METHOD=SKIP` to the option list (the result of the function will be a multi-polygon with all polygons as toplevel polygons) or only make it analyze counterclockwise polygons by adding `METHOD=ONLY_CCW` to the option list if you can assume that the outline of holes is counterclockwise defined (this is the convention for shapefiles e.g.)

If the `OGR_ORGANIZE_POLYGONS` configuration option is defined, its value will override the value of the `METHOD` option of `papszOptions` (usefull to modify the behaviour of the shapefile driver)

#### Parameters

<i>papoPolygons</i>	array of geometry pointers - should all be OGRPolygons. Ownership of the geometries is passed, but not of the array itself.
<i>nPolygonCount</i>	number of items in <code>papoPolygons</code>
<i>pblsValid-Geometry</i>	value will be set TRUE if result is valid or FALSE otherwise.
<i>papszOptions</i>	a list of strings for passing options

#### Returns

a single resulting geometry (either **OGRPolygon** (p. ??) or **OGRMultiPolygon** (p. ??)).

References `OGRPolygon::addRing()`, `CPLDebug()`, `CPLError()`, `CPLGetConfigOption()`, `CSLTestBoolean()`, `OGRPolygon::exportToWkt()`, `OGRPolygon::get_Area()`, `OGRGeometry::getEnvelope()`, `OGRPolygon::getExterior-Ring()`, `OGRLineString::getNumPoints()`, `OGRLineString::getPoint()`, `haveGEOS()`, `OGRLinearRing::isClockwise()`, `OGRGeometry::Overlaps()`, and `wkbPolygon`.

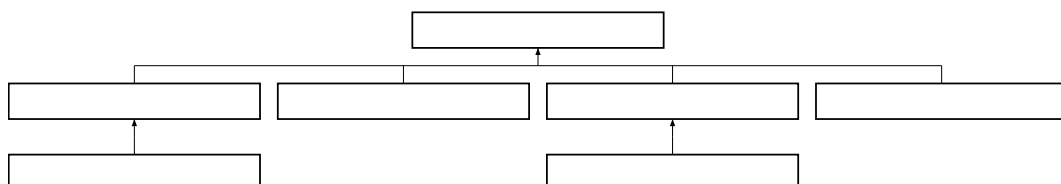
The documentation for this class was generated from the following files:

- `ogr_geometry.h`
- `ogrgeometryfactory.cpp`

## 12.59 OGRLayer Class Reference

```
#include <ogr_sfc_frmts.h>
```

Inheritance diagram for OGRLayer:



#### 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 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 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 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 **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 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)

*Intersection of two layers.*
- OGRErr **Union** (**OGRLayer** \*pLayerMethod, **OGRLayer** \*pLayerResult, char \*\*papszOptions=NULL, GDALProgressFunc pfnProgress=NULL, void \*pProgressArg=NULL)

*Union of two layers.*
- OGRErr **SymDifference** (**OGRLayer** \*pLayerMethod, **OGRLayer** \*pLayerResult, char \*\*papszOptions, GDALProgressFunc pfnProgress, void \*pProgressArg)

*Symmetrical difference of two layers.*
- OGRErr **Identity** (**OGRLayer** \*pLayerMethod, **OGRLayer** \*pLayerResult, char \*\*papszOptions=NULL, GDALProgressFunc pfnProgress=NULL, void \*pProgressArg=NULL)

*Identify the features of this layer with the ones from the identity layer.*
- OGRErr **Update** (**OGRLayer** \*pLayerMethod, **OGRLayer** \*pLayerResult, char \*\*papszOptions=NULL, GDALProgressFunc pfnProgress=NULL, void \*pProgressArg=NULL)

*Update this layer with features from the update layer.*
- OGRErr **Clip** (**OGRLayer** \*pLayerMethod, **OGRLayer** \*pLayerResult, char \*\*papszOptions=NULL, GDALProgressFunc pfnProgress=NULL, void \*pProgressArg=NULL)

*Clip off areas that are not covered by the method layer.*
- OGRErr **Erase** (**OGRLayer** \*pLayerMethod, **OGRLayer** \*pLayerResult, char \*\*papszOptions=NULL, GDALProgressFunc pfnProgress=NULL, void \*pProgressArg=NULL)

*Remove areas that are covered by the method layer.*
- int **Reference** ()

*Increment layer reference count.*
- int **Dereference** ()

*Decrement layer reference count.*
- int **GetRefCount** () const

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

*Reorder an existing field on a layer.*

### 12.59.1 Detailed Description

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

### 12.59.2 Member Function Documentation

#### 12.59.2.1 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** (p. ??) for the layer will be updated to reflect the altered field. Applications should never modify the **OGRFeatureDefn** (p. ??) 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 `OLCAAlterFieldDefn` 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()` (p. ??).

#### Parameters

<i>iField</i>	index of the field whose definition must be altered.
<i>poNewFieldDefn</i>	new field definition
<i>nFlags</i>	combination of <code>ALTER_NAME_FLAG</code> , <code>ALTER_TYPE_FLAG</code> and <code>ALTER_WIDTH_PRECISION_FLAG</code> 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

Reimplemented in `OGRProxiedLayer` (p. ??), and `OGRLayerDecorator` (p. ??).

References `CPL::Error()`.

Referenced by `OGRLayerDecorator::AlterFieldDefn()`, and `OGRProxiedLayer::AlterFieldDefn()`.

**12.59.2.2** `OGR::OGRLayer::Clip ( OGR::OGRLayer * pLayerMethod, OGR::OGRLayer * pLayerResult, char ** ppszOptions = NULL, GDALProgressFunc pfnProgress = NULL, void * pProgressArg = NULL )`

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.
- `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()` (p. ??).

## Parameters

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

## Since

OGR 1.10

References OGRGeometry::clone(), CPLError(), CreateFeature(), CSLTestBoolean(), GetFeatureCount(), OGRFeature::GetGeometryRef(), GetLayerDefn(), GetNextFeature(), OGRGeometryFactory::haveGEOS(), OGRGeometry::Intersection(), OGRGeometry::IsEmpty(), ResetReading(), OGRFeature::SetFieldsFrom(), OGRFeature::SetGeometryDirectly(), SetSpatialFilter(), and OGRGeometry::Union().

### 12.59.2.3 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()** (p. ??).

## Parameters

<i>poFeature</i>	the feature to write to disk.
------------------	-------------------------------

## Returns

OGRERR\_NONE on success.

Reimplemented in **OGRUnionLayer** (p. ??), **OGRProxiedLayer** (p. ??), **OGRWarpedLayer** (p. ??), and **OGR-LayerDecorator** (p. ??).

Referenced by Clip(), OGRDataSource::CopyLayer(), OGRLayerDecorator::CreateFeature(), OGRWarpedLayer::CreateFeature(), OGRProxiedLayer::CreateFeature(), OGRUnionLayer::CreateFeature(), Erase(), Identity(), Intersection(), SymDifference(), Union(), and Update().

### 12.59.2.4 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** (p. ??) for the layer will be updated to reflect the new field. Applications should never modify the **OGRFeatureDefn** (p. ??) 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 existings features of the backing file/database should be updated accordingly.



This function is the same as the C function **OGR\_L\_CreateField()** (p. ??).

## Parameters

<i>poField</i>	field definition to write to disk.
<i>bApproxOK</i>	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 **OGRProxiedLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

References CPLError().

Referenced by OGRDataSource::CopyLayer(), OGRLayerDecorator::CreateField(), and OGRProxiedLayer::CreateField().

#### 12.59.2.5 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()** (p. ??) 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()** (p. ??).

## Parameters

<i>nFID</i>	the feature id to be deleted from the layer
-------------	---

## Returns

OGRERR\_NONE on success.

Reimplemented in **OGRProxiedLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

Referenced by OGRLayerDecorator::DeleteFeature(), and OGRProxiedLayer::DeleteFeature().

#### 12.59.2.6 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** (p. ??) for the layer will be updated to reflect the deleted field. Applications should never modify the **OGRFeatureDefn** (p. ??) 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 existings features of the backing file/database should be updated accordingly.

This function is the same as the C function **OGR\_L\_DeleteField()** (p. ??).

## Parameters

<i>iField</i>	index of the field to delete.
---------------	-------------------------------

**Returns**

OGRERR\_NONE on success.

**Since**

OGR 1.9.0

Reimplemented in **OGRProxiedLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

References CPLError().

Referenced by OGRLayerDecorator::DeleteField(), and OGRProxiedLayer::DeleteField().

**12.59.2.7 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.

**12.59.2.8 OGRErr OGRLayer::Erase ( OGRLayer \* *pLayerMethod*, OGRLayer \* *pLayerResult*, char \*\* *papszOptions* = NULL, GDALProgressFunc *pfnProgress* = NULL, void \* *pProgressArg* = NULL )**

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.
- 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()** (p. ??).

## Parameters

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

## Since

OGR 1.10

References OGRGeometry::clone(), CPLError(), CreateFeature(), CSLTestBoolean(), OGRGeometry::Difference(), GetFeatureCount(), OGRFeature::GetGeometryRef(), GetLayerDefn(), GetNextFeature(), OGRGeometryFactory::haveGEOS(), OGRGeometry::IsEmpty(), ResetReading(), OGRFeature::SetFieldsFrom(), OGRFeature::SetGeometryDirectly(), SetSpatialFilter(), and OGRGeometry::Union().

### 12.59.2.9 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()** (p. ??) 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()** (p. ??).

## Parameters

<i>psExtent</i>	the structure in which the extent value will be returned.
<i>bForce</i>	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 **OGRUnionLayer** (p. ??), **OGRProxiedLayer** (p. ??), **OGRGenSQLResultsLayer** (p. ??), **OGRWarpedLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

References OGRGeometry::getEnvelope(), OGRFeature::GetGeometryRef(), GetGeomType(), GetLayerDefn(), GetNextFeature(), OGRGeometry::IsEmpty(), ResetReading(), and wkbNone.

Referenced by OGRLayerDecorator::GetExtent(), OGRWarpedLayer::GetExtent(), OGRGenSQLResultsLayer::GetExtent(), OGRProxiedLayer::GetExtent(), and Intersection().

### 12.59.2.10 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.

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

Use OGRLayer::TestCapability(OLCRandomRead) to establish if this layer supports efficient random access reading via **GetFeature()** (p. ??); 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 are generally considered interrupted by a **GetFeature()** (p. ??) call.

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

This method is the same as the C function **OGR\_L\_GetFeature()** (p. ??).

#### Parameters

<i>nFID</i>	the feature id of the feature to read.
-------------	--

#### Returns

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

Reimplemented in **OGRUnionLayer** (p. ??), **OGRProxiedLayer** (p. ??), **OGRGenSQLResultsLayer** (p. ??), **OGRWarpedLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

References OGRFeature::GetFID(), GetNextFeature(), and ResetReading().

Referenced by OGRLayerDecorator::GetFeature(), OGRWarpedLayer::GetFeature(), OGRGenSQLResultsLayer::GetFeature(), OGRProxiedLayer::GetFeature(), and OGRUnionLayer::GetFeature().

#### 12.59.2.11 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()** (p. ??).

#### Parameters

<i>bForce</i>	Flag indicating whether the count should be computed even if it is expensive.
---------------	---

#### Returns

feature count, -1 if count not known.

Reimplemented in **OGRUnionLayer** (p. ??), **OGRProxiedLayer** (p. ??), **OGRGenSQLResultsLayer** (p. ??), **OGRWarpedLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

References GetNextFeature(), and ResetReading().

Referenced by Clip(), Erase(), OGRLayerDecorator::GetFeatureCount(), OGRWarpedLayer::GetFeatureCount(), OGRGenSQLResultsLayer::GetFeatureCount(), OGRProxiedLayer::GetFeatureCount(), OGRUnionLayer::GetFeatureCount(), Identity(), Intersection(), SymDifference(), Union(), and Update().

### 12.59.2.12 `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()** (p. ??).

#### Returns

fid column name.

Reimplemented in **OGRProxiedLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

Referenced by **OGRLayerDecorator::GetFIDColumn()**, and **OGRProxiedLayer::GetFIDColumn()**.

### 12.59.2.13 `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()** (p. ??).

#### Returns

geometry column name.

Reimplemented in **OGRProxiedLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

Referenced by **OGRLayerDecorator::GetGeometryColumn()**, and **OGRProxiedLayer::GetGeometryColumn()**.

### 12.59.2.14 `OGRwkbGeometryType OGRLayer::GetGeomType ( ) [virtual]`

Return the layer geometry type.

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

This method is the same as the C function **OGR\_L\_GetGeomType()** (p. ??).

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

#### Returns

the geometry type

#### Since

OGR 1.8.0

Reimplemented in **OGRProxiedLayer** (p. ??), **OGRUnionLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

References **OGRFeatureDefn::GetGeomType()**, and **GetLayerDefn()**.

Referenced by **GetExtent()**, **OGRLayerDecorator::GetGeomType()**, **OGRUnionLayer::GetGeomType()**, and **OGRProxiedLayer::GetGeomType()**.

### 12.59.2.15 `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()** (p. ??) call.

#### Parameters

<i>pszTag</i>	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

##### 12.59.2.16 OGRFeatureDefn \* OGRLayer::GetLayerDefn ( ) [pure virtual]

Fetch the schema information for this layer.

The returned **OGRFeatureDefn** (p. ??) is owned by the **OGRLayer** (p. ??), 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()** (p. ??).

#### Returns

feature definition.

Implemented in **OGRUnionLayer** (p. ??), **OGRProxiedLayer** (p. ??), **OGRGenSQLResultsLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

Referenced by Clip(), OGRSFDriver::CopyDataSource(), OGRDataSource::CopyLayer(), Erase(), GetExtent(), GetGeomType(), OGRLayerDecorator::GetLayerDefn(), OGRProxiedLayer::GetLayerDefn(), OGRUnionLayer::GetLayerDefn(), GetName(), Identity(), Intersection(), ReorderField(), SetAttributeFilter(), SetIgnoredFields(), SymDifference(), Union(), and Update().

##### 12.59.2.17 const char \* OGRLayer::GetName ( ) [virtual]

Return the layer name.

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

This method is the same as the C function **OGR\_L\_GetName()** (p. ??).

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

#### Returns

the layer name (must not be freed)

#### Since

OGR 1.8.0

Reimplemented in **OGRProxiedLayer** (p. ??), **OGRUnionLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

References GetLayerDefn(), and OGRFeatureDefn::GetName().

Referenced by OGRDataSource::GetLayerByName(), OGRLayerDecorator::GetName(), and OGRProxiedLayer::GetName().

### 12.59.2.18 **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()** (p. ??). It is critical that all features associated with an **OGRLayer** (p. ??) (more specifically an **OGRFeatureDefn** (p. ??)) be deleted before that layer/datasource is deleted.

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

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

This method is the same as the C function **OGR\_L\_GetNextFeature()** (p. ??).

#### Returns

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

Implemented in **OGRUnionLayer** (p. ??), **OGRProxiedLayer** (p. ??), **OGRGenSQLResultsLayer** (p. ??), **OGRWarpedLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

Referenced by **Clip()**, **OGRDataSource::CopyLayer()**, **Erase()**, **GetExtent()**, **GetFeature()**, **GetFeatureCount()**, **OGRLayerDecorator::GetNextFeature()**, **OGRWarpedLayer::GetNextFeature()**, **OGRGenSQLResultsLayer::GetNextFeature()**, **OGRProxiedLayer::GetNextFeature()**, **OGRUnionLayer::GetNextFeature()**, **Identity()**, **Intersection()**, **SetNextByIndex()**, **SymDifference()**, **Union()**, and **Update()**.

### 12.59.2.19 **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.

Referenced by **OGRDataSource::GetSummaryRefCount()**.

### 12.59.2.20 **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()** (p. ??).

#### Returns

spatial filter geometry.

Reimplemented in **OGRProxiedLayer** (p. ??), **OGRGenSQLResultsLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

Referenced by **OGRLayerDecorator::GetSpatialFilter()**, and **OGRProxiedLayer::GetSpatialFilter()**.

### 12.59.2.21 **OGRSpatialReference \* OGRLayer::GetSpatialRef ( )** [inline], [virtual]

Fetch the spatial reference system for this layer.

The returned object is owned by the **OGRLayer** (p. ??) and should not be modified or freed by the application.

This method is the same as the C function **OGR\_L\_GetSpatialRef()** (p. ??).



**Returns**

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

Reimplemented in **OGRUnionLayer** (p. ??), **OGRProxiedLayer** (p. ??), **OGRGenSQLResultsLayer** (p. ??), **OGRWarpedLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

Referenced by OGRDataSource::CopyLayer(), OGRLayerDecorator::GetSpatialRef(), OGRGenSQLResultsLayer::GetSpatialRef(), OGRProxiedLayer::GetSpatialRef(), and OGRUnionLayer::GetSpatialRef().

**12.59.2.22 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 **OGRProxiedLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

Referenced by OGRLayerDecorator::GetStyleTable(), and OGRProxiedLayer::GetStyleTable().

**12.59.2.23 OGRErr OGRLayer::Identity ( OGRLayer \* pLayerMethod, OGRLayer \* pLayerResult, char \*\* papszOptions = NULL, GDALProgressFunc pfnProgress = NULL, void \* pProgressArg = NULL )**

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.
- 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\_Identity()** (p. ??).

## Parameters

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

## Since

OGR 1.10

References OGRGeometry::clone(), CPLError(), CreateFeature(), CSLTestBoolean(), OGRGeometry::Difference(), OGRGeometry::getDimension(), GetFeatureCount(), GetLayerDefn(), GetNextFeature(), OGRGeometryFactory::haveGEOS(), OGRGeometry::Intersection(), OGRGeometry::IsEmpty(), ResetReading(), OGRFeature::SetFieldsFrom(), OGRFeature::SetGeometryDirectly(), and SetSpatialFilter().

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

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 it to YES to go on, even when a feature could not be inserted.
- 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\_Intersection()** (p. ??).

## Parameters

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

**Since**

OGR 1.10

References CPLError(), CreateFeature(), CSLTestBoolean(), OGRGeometry::getDimension(), OGRGeometry::getEnvelope(), GetExtent(), GetFeatureCount(), GetLayerDefn(), GetNextFeature(), OGRGeometryFactory::haveGEOS(), OGRGeometry::Intersection(), OGRGeometry::IsEmpty(), ResetReading(), OGRFeature::SetFieldsFrom(), OGRFeature::SetGeometryDirectly(), and SetSpatialFilter().

**12.59.2.25 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.

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

Reorder an existing field on a layer.

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

You must use this to reorder existing fields on a real layer. Internally the **OGRFeatureDefn** (p. ??) for the layer will be updated to reflect the reordering of the fields. Applications should never modify the **OGRFeatureDefn** (p. ??) 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 existings features of the backing file/database should be updated accordingly.

This function is the same as the C function **OGR\_L\_ReorderField()** (p. ??).

## Parameters

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

References CPLError(), CPLMalloc(), OGRFeatureDefn::GetFieldCount(), GetLayerDefn(), and ReorderFields().

**12.59.2.27** 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** (p. ??) for the layer will be updated to reflect the reordering of the fields. Applications should never modify the **OGRFeatureDefn** (p. ??) 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 existings features of the backing file/database should be updated accordingly.

This function is the same as the C function **OGR\_L\_ReorderFields()** (p. ??).

## Parameters

<i>panMap</i>	an array of <b>GetLayerDefn()</b> (p. ??)->GetFieldCount() elements which is a permutation of [0, <b>GetLayerDefn()</b> (p. ??)->GetFieldCount()-1].
---------------	--

## Returns

OGRERR\_NONE on success.

## Since

OGR 1.9.0

Reimplemented in **OGRProxiedLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

References CPLError().

Referenced by ReorderField(), OGRLayerDecorator::ReorderFields(), and OGRProxiedLayer::ReorderFields().

**12.59.2.28** void OGRLayer::ResetReading ( ) [pure virtual]

Reset feature reading to start on the first feature.

This affects **GetNextFeature()** (p. ??).

This method is the same as the C function **OGR\_L\_ResetReading()** (p. ??).

Implemented in **OGRUnionLayer** (p. ??), **OGRProxiedLayer** (p. ??), **OGRGenSQLResultsLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

Referenced by Clip(), OGRDataSource::CopyLayer(), Erase(), GetExtent(), GetFeature(), GetFeatureCount(), Identity(), Intersection(), OGRLayerDecorator::ResetReading(), OGRGenSQLResultsLayer::ResetReading(), OGRProxiedLayer::ResetReading(), SetAttributeFilter(), SetNextByIndex(), SetSpatialFilter(), SymDifference(), Union(), and Update().

#### 12.59.2.29 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()** (p. ??) 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()** (p. ??)).

This method is the same as the C function **OGR\_L\_SetAttributeFilter()** (p. ??).

##### Parameters

<i>pszQuery</i>	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 **OGRUnionLayer** (p. ??), **OGRProxiedLayer** (p. ??), **OGRGenSQLResultsLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

References GetLayerDefn(), and ResetReading().

Referenced by OGRGenSQLResultsLayer::ResetReading(), OGRLayerDecorator::SetAttributeFilter(), OGRGenSQLResultsLayer::SetAttributeFilter(), OGRProxiedLayer::SetAttributeFilter(), and OGRUnionLayer::SetAttributeFilter().

#### 12.59.2.30 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** (p. ??).

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

This method is the same as the C function **OGR\_L\_SetFeature()** (p. ??).

##### Parameters

<i>poFeature</i>	the feature to write.
------------------	-----------------------

**Returns**

OGRERR\_NONE if the operation works, otherwise an appropriate error code.

Reimplemented in **OGRUnionLayer** (p. ??), **OGRProxiedLayer** (p. ??), **OGRWarpedLayer** (p. ??), and **OGR-LayerDecorator** (p. ??).

Referenced by OGRLayerDecorator::SetFeature(), OGRWarpedLayer::SetFeature(), OGRProxiedLayer::SetFeature(), and OGRUnionLayer::SetFeature().

### 12.59.2.31 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()** (p. ??) / **GetNextFeature()** (p. ??) 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()** (p. ??)

**Parameters**

<i>papszFields</i>	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 **OGRProxiedLayer** (p. ??), **OGRUnionLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

References OGRFeatureDefn::GetFieldCount(), OGRFeatureDefn::GetFieldDefn(), OGRFeatureDefn::GetFieldIndex(), GetLayerDefn(), OGRFeatureDefn::SetGeometryIgnored(), OGRFieldDefn::SetIgnored(), and OGRFeatureDefn::SetStyleIgnored().

Referenced by OGRLayerDecorator::SetIgnoredFields(), OGRUnionLayer::SetIgnoredFields(), and OGRProxiedLayer::SetIgnoredFields().

### 12.59.2.32 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()** (p. ??) 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()** (p. ??) 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()** (p. ??) efficiently implemented. In all other cases the default implementation which calls **ResetReading()** (p. ??) and then calls **GetNextFeature()** (p. ??) nIndex times is used. To determine if fast seeking is available on the current layer use the **TestCapability()** (p. ??) method with a value of OLCFastSetNextByIndex.

This method is the same as the C function **OGR\_L\_SetNextByIndex()** (p. ??).

## Parameters

<i>nIndex</i>	the index indicating how many steps into the result set to seek.
---------------	--

## Returns

OGRERR\_NONE on success or an error code.

Reimplemented in **OGRProxiedLayer** (p. ??), **OGRGenSQLResultsLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

References `GetNextFeature()`, and `ResetReading()`.

Referenced by `OGRLayerDecorator::SetNextByIndex()`, `OGRGenSQLResultsLayer::SetNextByIndex()`, and `OGRProxiedLayer::SetNextByIndex()`.

### 12.59.2.33 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()** (p. ??) 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()** (p. ??)) 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()** (p. ??)). In the future this may be generalized.

This method is the same as the C function **OGR\_L\_SetSpatialFilter()** (p. ??).

## Parameters

<i>poFilter</i>	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 **OGRUnionLayer** (p. ??), **OGRProxiedLayer** (p. ??), **OGRGenSQLResultsLayer** (p. ??), **OGRWarpedLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

References `ResetReading()`.

Referenced by `Clip()`, `Erase()`, `OGRUnionLayer::GetFeatureCount()`, `Identity()`, `Intersection()`, `OGRGenSQLResultsLayer::ResetReading()`, `OGRLayerDecorator::SetSpatialFilter()`, `OGRWarpedLayer::SetSpatialFilter()`, `OGRGenSQLResultsLayer::SetSpatialFilter()`, `OGRProxiedLayer::SetSpatialFilter()`, `OGRUnionLayer::SetSpatialFilter()`, `SetSpatialFilterRect()`, `SymDifference()`, `OGRUnionLayer::TestCapability()`, `Union()`, and `Update()`.

### 12.59.2.34 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()** (p. ??) 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()** (p. ??)). Internally this method is normally implemented as creating a 5 vertex closed rectangular polygon and passing it to **OGRLayer::SetSpatialFilter()** (p. ??). 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()** (p. ??).

## Parameters

<i>dfMinX</i>	the minimum X coordinate for the rectangular region.
<i>dfMinY</i>	the minimum Y coordinate for the rectangular region.
<i>dfMaxX</i>	the maximum X coordinate for the rectangular region.
<i>dfMaxY</i>	the maximum Y coordinate for the rectangular region.

Reimplemented in **OGRWarpedLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

References OGRLineString::addPoint(), OGRPolygon::addRing(), and SetSpatialFilter().

Referenced by OGRWarpedLayer::SetSpatialFilter(), OGRLayerDecorator::SetSpatialFilterRect(), and OGRWarpedLayer::SetSpatialFilterRect().

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

Set layer style table.

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

This method is the same as the C function OGR\_L\_SetStyleTable().

## Parameters

<i>poStyleTable</i>	pointer to style table to set
---------------------	-------------------------------

Reimplemented in **OGRProxiedLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

References OGRStyleTable::Clone().

Referenced by OGRLayerDecorator::SetStyleTable(), and OGRProxiedLayer::SetStyleTable().

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

Set layer style table.

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

This method is the same as the C function OGR\_L\_SetStyleTableDirectly().

## Parameters

<i>poStyleTable</i>	pointer to style table to set
---------------------	-------------------------------

Reimplemented in **OGRProxiedLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

Referenced by OGRLayerDecorator::SetStyleTableDirectly(), and OGRProxiedLayer::SetStyleTableDirectly().

**12.59.2.37** 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.
- 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()** (p. ??).

**Parameters**

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

**Since**

OGR 1.10

References OGRGeometry::clone(), CPLError(), CreateFeature(), CSLTestBoolean(), OGRGeometry::Difference(), GetFeatureCount(), GetLayerDefn(), GetNextFeature(), OGRGeometryFactory::haveGEOS(), OGRGeometry::IsEmpty(), ResetReading(), OGRFeature::SetFieldsFrom(), OGRFeature::SetGeometryDirectly(), and SetSpatialFilter().

**12.59.2.38 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()** (p. ??) that will ensure all data is correctly flushed.

This method is the same as the C function **OGR\_L\_SyncToDisk()** (p. ??).

## Returns

OGRERR\_NONE if no error occurs (even if nothing is done) or an error code.

Reimplemented in **OGRProxiedLayer** (p. ??), **OGRUnionLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

Referenced by OGRLayerDecorator::SyncToDisk(), OGRUnionLayer::SyncToDisk(), OGRProxiedLayer::SyncToDisk(), and OGRDataSource::SyncToDisk().

**12.59.2.39** `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()** (p. ??) method is implemented in an optimized way for this layer, as opposed to the default implementation using **ResetReading()** (p. ??) and **GetNextFeature()** (p. ??) to find the requested feature id.
- **OLCSequentialWrite** / "SequentialWrite": TRUE if the **CreateFeature()** (p. ??) method works for this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. ??) class may returned FALSE for other layer instances that are effectively read-only.
- **OLCRandomWrite** / "RandomWrite": TRUE if the **SetFeature()** (p. ??) method is operational on this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. ??) 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** (p. ??) 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()** (p. ??)) 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()** (p. ??)) 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()** (p. ??) call efficiently, otherwise FALSE.
- **OLCCreateField** / "CreateField": TRUE if this layer can create new fields on the current layer using **CreateField()** (p. ??), otherwise FALSE.
- **OLCDeleteField** / "DeleteField": TRUE if this layer can delete existing fields on the current layer using **DeleteField()** (p. ??), otherwise FALSE.
- **OLCReorderFields** / "ReorderFields": TRUE if this layer can reorder existing fields on the current layer using **ReorderField()** (p. ??) or **ReorderFields()** (p. ??), otherwise FALSE.
- **OLCAlterFieldDefn** / "AlterFieldDefn": TRUE if this layer can alter the definition of an existing field on the current layer using **AlterFieldDefn()** (p. ??), otherwise FALSE.
- **OLCDeleteFeature** / "DeleteFeature": TRUE if the **DeleteFeature()** (p. ??) 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() (p. ??) method.

This method is the same as the C function **OGR\_L\_TestCapability()** (p. ??).

#### Parameters

<i>pszCap</i>	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.

Implemented in **OGRUnionLayer** (p. ??), **OGRProxiedLayer** (p. ??), **OGRGenSQLResultsLayer** (p. ??), **OGRWarpedLayer** (p. ??), and **OGRLayerDecorator** (p. ??).

Referenced by OGRDataSource::CopyLayer(), OGRLayerDecorator::TestCapability(), OGRWarpedLayer::TestCapability(), OGRGenSQLResultsLayer::TestCapability(), and OGRProxiedLayer::TestCapability().

**12.59.2.40 OGRErr OGRLayer::Union ( OGRLayer \* pLayerMethod, OGRLayer \* pLayerResult, char \*\* papszOptions = NULL, GDALProgressFunc pfnProgress = NULL, void \* pProgressArg = NULL )**

Union 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. 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.
- **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\_Union()** (p. ??).

## Parameters

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

## Since

OGR 1.10

References OGRGeometry::clone(), CPLError(), CreateFeature(), CSLTestBoolean(), OGRGeometry::Difference(), OGRGeometry::getDimension(), GetFeatureCount(), GetLayerDefn(), GetNextFeature(), OGRGeometryFactory::haveGEOS(), OGRGeometry::Intersection(), OGRGeometry::IsEmpty(), ResetReading(), OGRFeature::SetFieldsFrom(), OGRFeature::SetGeometryDirectly(), and SetSpatialFilter().

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

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 areas 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 that 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.
- 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()** (p. ??).

## Parameters

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

## Since

OGR 1.10

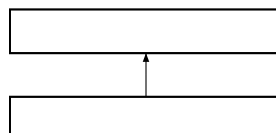
References OGRGeometry::clone(), CPLError(), CreateFeature(), CSLTestBoolean(), OGRGeometry::Difference(), GetFeatureCount(), GetLayerDefn(), GetNextFeature(), OGRGeometryFactory::haveGEOS(), OGRGeometry::IsEmpty(), ResetReading(), OGRFeature::SetFieldsFrom(), OGRFeature::SetGeometry(), OGRFeature::SetGeometryDirectly(), and SetSpatialFilter().

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

- **ogrsf\_frmts.h**
- ogrsf\_frmts.dox
- ogrlayer.cpp

## 12.60 OGRLayerAttrIndex Class Reference

Inheritance diagram for OGRLayerAttrIndex:

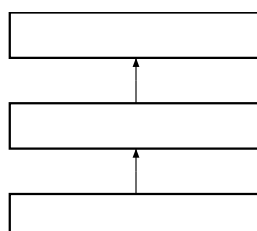


The documentation for this class was generated from the following files:

- ogr\_attrind.h
- ogr\_attrind.cpp

## 12.61 OGRLayerDecorator Class Reference

Inheritance diagram for OGRLayerDecorator:



## 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 OGRErr **SetAttributeFilter** (const char \*)  
*Set a new attribute query.*
- virtual void **ResetReading** ()  
*Reset feature reading to start on the first feature.*
- virtual **OGRFeature \*** **GetNextFeature** ()  
*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** ()  
*Fetch the schema information for this 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 int **TestCapability** (const char \*)  
*Test if this layer supported the named capability.*
- 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 **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 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.*

## 12.61.1 Member Function Documentation

### 12.61.1.1 OGRErr OGRLayerDecorator::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** (p. ??) for the layer will be updated to reflect the altered field. Applications should never modify the **OGRFeatureDefn** (p. ??) 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 **OLCAAlterFieldDefn** 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()** (p. ??).

#### Parameters

<i>iField</i>	index of the field whose definition must be altered.
<i>poNewFieldDefn</i>	new field definition
<i>nFlags</i>	combination of ALTER_NAME_FLAG, ALTER_TYPE_FLAG and ALTER_WIDTH_PRECISION_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

Reimplemented from **OGRLayer** (p. ??).

References **OGRLayer::AlterFieldDefn()**.

### 12.61.1.2 OGRErr OGRLayerDecorator::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()** (p. ??).



## Parameters

<i>poFeature</i>	the feature to write to disk.
------------------	-------------------------------

## Returns

OGRERR\_NONE on success.

Reimplemented from **OGRLayer** (p. ??).

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

References OGRLayer::CreateFeature().

### 12.61.1.3 OGRErr OGRLayerDecorator::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** (p. ??) for the layer will be updated to reflect the new field. Applications should never modify the **OGRFeatureDefn** (p. ??) 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 existings features of the backing file/database should be updated accordingly.

This function is the same as the C function **OGR\_L\_CreateField()** (p. ??).

## Parameters

<i>poField</i>	field definition to write to disk.
<i>bApproxOK</i>	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 from **OGRLayer** (p. ??).

References OGRLayer::CreateField().

### 12.61.1.4 OGRErr OGRLayerDecorator::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()** (p. ??) 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()** (p. ??).

## Parameters

<i>nFID</i>	the feature id to be deleted from the layer
-------------	---

## Returns

OGRERR\_NONE on success.

Reimplemented from **OGRLayer** (p. ??).

References OGRLayer::DeleteFeature().

### 12.61.1.5 OGRErr OGRLayerDecorator::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** (p. ??) for the layer will be updated to reflect the deleted field. Applications should never modify the **OGRFeatureDefn** (p. ??) 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()** (p. ??).

#### Parameters

<i>iField</i>	index of the field to delete.
---------------	-------------------------------

#### Returns

OGRERR\_NONE on success.

#### Since

OGR 1.9.0

Reimplemented from **OGRLayer** (p. ??).

References **OGRLayer::DeleteField()**.

### 12.61.1.6 OGRErr OGRLayerDecorator::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 known. 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()** (p. ??) 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()** (p. ??).

#### Parameters

<i>psExtent</i>	the structure in which the extent value will be returned.
<i>bForce</i>	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 from **OGRLayer** (p. ??).

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

References **OGRLayer::GetExtent()**.

**12.61.1.7 OGRFeature \* OGRLayerDecorator::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.

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

Use OGRLayer::TestCapability(OLCRandomRead) to establish if this layer supports efficient random access reading via **GetFeature()** (p. ??); 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 are generally considered interrupted by a **GetFeature()** (p. ??) call.

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

This method is the same as the C function **OGR\_L\_GetFeature()** (p. ??).

**Parameters**

<i>nFID</i>	the feature id of the feature to read.
-------------	--

**Returns**

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

Reimplemented from **OGRLayer** (p. ??).

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

References OGRLayer::GetFeature().

**12.61.1.8 int OGRLayerDecorator::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()** (p. ??).

**Parameters**

<i>bForce</i>	Flag indicating whether the count should be computed even if it is expensive.
---------------	---

**Returns**

feature count, -1 if count not known.

Reimplemented from **OGRLayer** (p. ??).

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

References OGRLayer::GetFeatureCount().

**12.61.1.9 const char \* OGRLayerDecorator::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()** (p. ??).

#### Returns

fid column name.

Reimplemented from **OGRLayer** (p. ??).

References **OGRLayer::GetFIDColumn()**.

#### 12.61.1.10 `const char * OGRLayerDecorator::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()** (p. ??).

#### Returns

geometry column name.

Reimplemented from **OGRLayer** (p. ??).

References **OGRLayer::GetGeometryColumn()**.

#### 12.61.1.11 `OGRwkbGeometryType OGRLayerDecorator::GetGeomType ( ) [virtual]`

Return the layer geometry type.

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

This method is the same as the C function **OGR\_L\_GetGeomType()** (p. ??).

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

#### Returns

the geometry type

#### Since

OGR 1.8.0

Reimplemented from **OGRLayer** (p. ??).

References **OGRLayer::GetGeomType()**.

#### 12.61.1.12 `OGRFeatureDefn * OGRLayerDecorator::GetLayerDefn ( ) [virtual]`

Fetch the schema information for this layer.

The returned **OGRFeatureDefn** (p. ??) is owned by the **OGRLayer** (p. ??), 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()** (p. ??).

#### Returns

feature definition.

Implements **OGRLayer** (p. ??).

References **OGRLayer::GetLayerDefn()**.

### 12.61.1.13 `const char * OGRLayerDecorator::GetName ( ) [virtual]`

Return the layer name.

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

This method is the same as the C function **OGR\_L\_GetName()** (p. ??).

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

#### Returns

the layer name (must not be freed)

#### Since

OGR 1.8.0

Reimplemented from **OGRLayer** (p. ??).

References **OGRLayer::GetName()**.

### 12.61.1.14 `OGRFeature * OGRLayerDecorator::GetNextFeature ( ) [virtual]`

Fetch the next available feature from this layer.

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

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

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

This method is the same as the C function **OGR\_L\_GetNextFeature()** (p. ??).

#### Returns

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

Implements **OGRLayer** (p. ??).

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

References **OGRLayer::GetNextFeature()**.

### 12.61.1.15 `OGRGeometry * OGRLayerDecorator::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()** (p. ??).

#### Returns

spatial filter geometry.

Reimplemented from **OGRLayer** (p. ??).

References **OGRLayer::GetSpatialFilter()**.

#### 12.61.1.16 **OGRSpatialReference** \* **OGRLayerDecorator::GetSpatialRef**( ) [virtual]

Fetch the spatial reference system for this layer.

The returned object is owned by the **OGRLayer** (p. ??) and should not be modified or freed by the application.

This method is the same as the C function **OGR\_L\_GetSpatialRef**() (p. ??).

##### Returns

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

Reimplemented from **OGRLayer** (p. ??).

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

References **OGRLayer::GetSpatialRef**().

#### 12.61.1.17 **OGRStyleTable** \* **OGRLayerDecorator::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 from **OGRLayer** (p. ??).

References **OGRLayer::GetStyleTable**().

#### 12.61.1.18 **OGRErr** **OGRLayerDecorator::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** (p. ??) for the layer will be updated to reflect the reordering of the fields. Applications should never modify the **OGRFeatureDefn** (p. ??) 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 existings features of the backing file/database should be updated accordingly.

This function is the same as the C function **OGR\_L\_ReorderFields**() (p. ??).

##### Parameters

<i>panMap</i>	an array of <b>GetLayerDefn</b> () (p. ??)-> <b>GetFieldCount</b> () elements which is a permutation of [0, <b>GetLayerDefn</b> () (p. ??)-> <b>GetFieldCount</b> ()-1].
---------------	--

**Returns**

OGRERR\_NONE on success.

**Since**

OGR 1.9.0

Reimplemented from **OGRLayer** (p. ??).

References OGRLayer::ReorderFields().

### 12.61.1.19 void OGRLayerDecorator::ResetReading( ) [virtual]

Reset feature reading to start on the first feature.

This affects **GetNextFeature()** (p. ??).

This method is the same as the C function **OGR\_L\_ResetReading()** (p. ??).

Implements **OGRLayer** (p. ??).

References OGRLayer::ResetReading().

### 12.61.1.20 OGRErr OGRLayerDecorator::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()** (p. ??) 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()** (p. ??)).

This method is the same as the C function **OGR\_L\_SetAttributeFilter()** (p. ??).

**Parameters**

<i>pszQuery</i>	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 from **OGRLayer** (p. ??).

References OGRLayer::SetAttributeFilter().

### 12.61.1.21 OGRErr OGRLayerDecorator::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** (p. ??).

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

This method is the same as the C function **OGR\_L\_SetFeature()** (p. ??).



## Parameters

<i>poFeature</i>	the feature to write.
------------------	-----------------------

## Returns

OGRERR\_NONE if the operation works, otherwise an appropriate error code.

Reimplemented from **OGRLayer** (p. ??).

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

References OGRLayer::SetFeature().

#### 12.61.1.22 OGRErr OGRLayerDecorator::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()** (p. ??) / **GetNextFeature()** (p. ??) 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()** (p. ??)

## Parameters

<i>papszFields</i>	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 from **OGRLayer** (p. ??).

References OGRLayer::SetIgnoredFields().

#### 12.61.1.23 OGRErr OGRLayerDecorator::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()** (p. ??) 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()** (p. ??) 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()** (p. ??) efficiently implemented. In all other cases the default implementation which calls **ResetReading()** (p. ??) and then calls **GetNextFeature()** (p. ??) nIndex times is used. To determine if fast seeking is available on the current layer use the **TestCapability()** (p. ??) method with a value of OLCFastSetNextByIndex.

This method is the same as the C function **OGR\_L\_SetNextByIndex()** (p. ??).

## Parameters

<i>nIndex</i>	the index indicating how many steps into the result set to seek.
---------------	--

#### Returns

OGRERR\_NONE on success or an error code.

Reimplemented from **OGRLayer** (p. ??).

References OGRLayer::SetNextByIndex().

#### 12.61.1.24 void OGRLayerDecorator::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()** (p. ??) 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()** (p. ??)) 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()** (p. ??)). In the future this may be generalized.

This method is the same as the C function **OGR\_L\_SetSpatialFilter()** (p. ??).

#### Parameters

<i>poFilter</i>	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 from **OGRLayer** (p. ??).

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

References OGRLayer::SetSpatialFilter().

#### 12.61.1.25 void OGRLayerDecorator::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()** (p. ??) 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()** (p. ??)). Internally this method is normally implemented as creating a 5 vertex closed rectangular polygon and passing it to **OGRLayer::SetSpatialFilter()** (p. ??). 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()** (p. ??).

#### Parameters

<i>dfMinX</i>	the minimum X coordinate for the rectangular region.
<i>dfMinY</i>	the minimum Y coordinate for the rectangular region.

<i>dfMaxX</i>	the maximum X coordinate for the rectangular region.
<i>dfMaxY</i>	the maximum Y coordinate for the rectangular region.

Reimplemented from **OGRLayer** (p. ??).

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

References OGRLayer::SetSpatialFilterRect().

#### 12.61.1.26 void OGRLayerDecorator::SetStyleTable ( OGRStyleTable \* *poStyleTable* ) [virtual]

Set layer style table.

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

This method is the same as the C function OGR\_L\_SetStyleTable().

##### Parameters

<i>poStyleTable</i>	pointer to style table to set
---------------------	-------------------------------

Reimplemented from **OGRLayer** (p. ??).

References OGRLayer::SetStyleTable().

#### 12.61.1.27 void OGRLayerDecorator::SetStyleTableDirectly ( OGRStyleTable \* *poStyleTable* ) [virtual]

Set layer style table.

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

This method is the same as the C function OGR\_L\_SetStyleTableDirectly().

##### Parameters

<i>poStyleTable</i>	pointer to style table to set
---------------------	-------------------------------

Reimplemented from **OGRLayer** (p. ??).

References OGRLayer::SetStyleTableDirectly().

#### 12.61.1.28 OGRErr OGRLayerDecorator::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()** (p. ??) that will ensure all data is correctly flushed.

This method is the same as the C function **OGR\_L\_SyncToDisk()** (p. ??).

##### Returns

OGRERR\_NONE if no error occurs (even if nothing is done) or an error code.

Reimplemented from **OGRLayer** (p. ??).

References OGRLayer::SyncToDisk().

12.61.1.29 `int OGRLayerDecorator::TestCapability ( const char * pszCap ) [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()** (p. ??) method is implemented in an optimized way for this layer, as opposed to the default implementation using **ResetReading()** (p. ??) and **GetNextFeature()** (p. ??) to find the requested feature id.
- **OLCSequentialWrite** / "SequentialWrite": TRUE if the **CreateFeature()** (p. ??) method works for this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. ??) class may returned FALSE for other layer instances that are effectively read-only.
- **OLCRandomWrite** / "RandomWrite": TRUE if the **SetFeature()** (p. ??) method is operational on this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. ??) 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** (p. ??) 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()** (p. ??)) 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()** (p. ??)) 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()** (p. ??) call efficiently, otherwise FALSE.
- **OLCCreateField** / "CreateField": TRUE if this layer can create new fields on the current layer using **CreateField()** (p. ??), otherwise FALSE.
- **OLCDeleteField** / "DeleteField": TRUE if this layer can delete existing fields on the current layer using **DeleteField()** (p. ??), otherwise FALSE.
- **OLCReorderFields** / "ReorderFields": TRUE if this layer can reorder existing fields on the current layer using **ReorderField()** (p. ??) or **ReorderFields()** (p. ??), otherwise FALSE.
- **OLCAlterFieldDefn** / "AlterFieldDefn": TRUE if this layer can alter the definition of an existing field on the current layer using **AlterFieldDefn()** (p. ??), otherwise FALSE.
- **OLCDeleteFeature** / "DeleteFeature": TRUE if the **DeleteFeature()** (p. ??) 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()** (p. ??) method.

This method is the same as the C function **OGR\_L\_TestCapability()** (p. ??).

## Parameters

<i>pszCap</i>	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.

Implements **OGRLayer** (p. ??).

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

References OGRLayer::TestCapability().

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

- ogrlayerdecorator.h
- ogrlayerdecorator.cpp

## 12.62 OGRLayerPool Class Reference

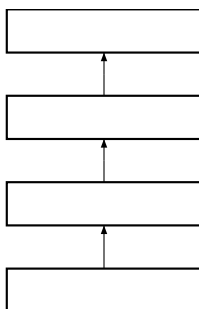
The documentation for this class was generated from the following files:

- ogrlayerpool.h
- ogrlayerpool.cpp

## 12.63 OGRLinearRing Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRLinearRing:



### Public Member Functions

- virtual const char \* **getGeometryName** () const  
*Fetch WKT name for geometry type.*
- virtual **OGRGeometry** \* **clone** () const  
*Make a copy of this object.*
- virtual int **isClockwise** () const  
*Returns TRUE if the ring has clockwise winding (or less than 2 points)*
- virtual void **closeRings** ()  
*Force rings to be closed.*

- virtual double **get\_Area** () const  
*Compute area of ring.*
- virtual int **WkbSize** () const  
*Returns size of related binary representation.*
- virtual OGRErr **importFromWkb** (unsigned char \*, int=-1)  
*Assign geometry from well known binary data.*
- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char \*) const  
*Convert a geometry into well known binary format.*

## Friends

- class **OGRPolygon**

### 12.63.1 Detailed Description

Concrete representation of a closed ring.

This class is functionally equivalent to an **OGRLineString** (p. ??), but has a separate identity to maintain alignment with the OpenGIS simple feature data model. It exists to serve as a component of an **OGRPolygon** (p. ??).

The **OGRLinearRing** (p. ??) has no corresponding free standing well known binary representation, so **importFromWkb()** (p. ??) and **exportToWkb()** (p. ??) will not actually work. There is a non-standard GDAL WKT representation though.

Because **OGRLinearRing** (p. ??) is not a "proper" free standing simple features object, it cannot be directly used on a feature via **SetGeometry()**, and cannot generally be used with GEOS for operations like **Intersects()** (p. ??). Instead the polygon should be used, or the **OGRLinearRing** (p. ??) should be converted to an **OGRLineString** (p. ??) for such operations.

### 12.63.2 Member Function Documentation

#### 12.63.2.1 OGRGeometry \* OGRLinearRing::clone ( ) const [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR\_G\_Clone()** (p. ??).

#### Returns

a new object instance with the same geometry, and spatial reference system as the original.

Reimplemented from **OGRLineString** (p. ??).

References **OGRGeometry::assignSpatialReference()**, **OGRGeometry::getSpatialReference()**, and **OGRLineString::setPoints()**.

#### 12.63.2.2 void OGRLinearRing::closeRings ( ) [virtual]

Force rings to be closed.

If this geometry, or any contained geometries has polygon rings that are not closed, they will be closed by adding the starting point at the end.

Reimplemented from **OGRGeometry** (p. ??).

References **OGRLineString::addPoint()**, **OGRLineString::getPoint()**, **OGRLineString::getX()**, **OGRLineString::getY()**, and **OGRLineString::getZ()**.

**12.63.2.3** `OGRERR OGRLinearRing::exportToWkb ( OGRwkbByteOrder eByteOrder, unsigned char * pabyData ) const` [virtual]

Convert a geometry into well known binary format.

This method relates to the SFCOM IWks::ExportToWKB() method.

This method is the same as the C function **OGR\_G\_ExportToWkb()** (p. ??).

#### Parameters

<i>eByteOrder</i>	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
<i>pabyData</i>	a buffer into which the binary representation is written. This buffer must be at least <b>OGR-Geometry::WkbSize()</b> (p. ??) byte in size.

#### Returns

Currently OGRERR\_NONE is always returned.

Reimplemented from **OGRLineString** (p. ??).

**12.63.2.4** `double OGRLinearRing::get_Area ( ) const` [virtual]

Compute area of ring.

The area is computed according to Green's Theorem:

Area is "Sum(x(i)\*(y(i+1) - y(i-1)))/2" for i = 0 to pointCount-1, assuming the last point is a duplicate of the first.

#### Returns

computed area.

Referenced by OGRPolygon::get\_Area().

**12.63.2.5** `const char * OGRLinearRing::getGeometryName ( ) const` [virtual]

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR\_G\_GetGeometryName()** (p. ??).

#### Returns

name used for this geometry type in well known text format. The returned pointer is to a static internal string and should not be modified or freed.

Reimplemented from **OGRLineString** (p. ??).

**12.63.2.6** `OGRERR OGRLinearRing::importFromWkb ( unsigned char * pabyData, int nSize = -1 )` [virtual]

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKB() method.

This method is the same as the C function **OGR\_G\_ImportFromWkb()** (p. ??).

## Parameters

<i>pabyData</i>	the binary input data.
<i>nSize</i>	the size of pabyData in bytes, or zero if not known.

## Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

Reimplemented from **OGRLineString** (p. ??).

**12.63.2.7** `int OGRLinearRing::isClockwise ( ) const [virtual]`

Returns TRUE if the ring has clockwise winding (or less than 2 points)

## Returns

TRUE if clockwise otherwise FALSE.

Referenced by OGRGeometryFactory::organizePolygons().

**12.63.2.8** `int OGRLinearRing::WkbSize ( ) const [virtual]`

Returns size of related binary representation.

This method returns the exact number of bytes required to hold the well known binary representation of this geometry object. Its computation may be slightly expensive for complex geometries.

This method relates to the SFCOM IWks::WkbSize() method.

This method is the same as the C function **OGR\_G\_WkbSize()** (p. ??).

## Returns

size of binary representation in bytes.

Reimplemented from **OGRLineString** (p. ??).

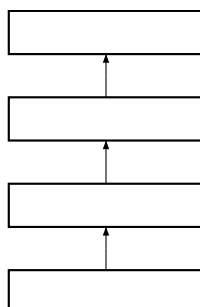
The documentation for this class was generated from the following files:

- **ogr\_geometry.h**
- **ogrlinearring.cpp**

## 12.64 OGRLineString Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRLineString:





## Public Member Functions

- **OGRLineString** ()  
*Create an empty line string.*
- virtual int **WkbSize** () const  
*Returns size of related binary representation.*
- virtual OGRErr **importFromWkb** (unsigned char \*, int=-1)  
*Assign geometry from well known binary data.*
- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char \*) const  
*Convert a geometry into well known binary format.*
- virtual OGRErr **importFromWkt** (char \*\*) *Assign geometry from well known text data.*
- virtual OGRErr **exportToWkt** (char \*\*ppszDstText) const  
*Convert a geometry into well known text format.*
- virtual int **getDimension** () const  
*Get the dimension of this object.*
- virtual **OGRGeometry** \* **clone** () const  
*Make a copy of this object.*
- virtual void **empty** ()  
*Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.*
- virtual void **getEnvelope** (**OGREnvelope** \*psEnvelope) const  
*Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.*
- virtual void **getEnvelope** (**OGREnvelope3D** \*psEnvelope) const  
*Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.*
- virtual OGRBoolean **IsEmpty** () const  
*Returns TRUE (non-zero) if the object has no points.*
- virtual double **get\_Length** () const  
*Returns the length of the curve.*
- virtual void **StartPoint** (**OGRPoint** \*) const  
*Return the curve start point.*
- virtual void **EndPoint** (**OGRPoint** \*) const  
*Return the curve end point.*
- virtual void **Value** (double, **OGRPoint** \*) const  
*Fetch point at given distance along curve.*
- int **getNumPoints** () const  
*Fetch vertex count.*
- void **getPoint** (int, **OGRPoint** \*) const  
*Fetch a point in line string.*
- double **getX** (int i) const  
*Get X at vertex.*
- double **getY** (int i) const  
*Get Y at vertex.*
- double **getZ** (int i) const  
*Get Z at vertex.*
- virtual OGRBoolean **Equals** (**OGRGeometry** \*) const  
*Returns TRUE if two geometries are equivalent.*
- virtual void **setCoordinateDimension** (int nDimension)  
*Set the coordinate dimension.*
- void **setNumPoints** (int)  
*Set number of points in geometry.*

- void **setPoint** (int, **OGRPoint** \*)  
*Set the location of a vertex in line string.*
- void **setPoint** (int, double, double, double)  
*Set the location of a vertex in line string.*
- void **setPoints** (int, **OGRRawPoint** \*, double \*=NULL)  
*Assign all points in a line string.*
- void **setPoints** (int, double \*padfX, double \*padfY, double \*padfZ=NULL)  
*Assign all points in a line string.*
- void **addPoint** (**OGRPoint** \*)  
*Add a point to a line string.*
- void **addPoint** (double, double, double)  
*Add a point to a line string.*
- void **getPoints** (**OGRRawPoint** \*, double \*=NULL) const  
*Returns all points of line string.*
- void **getPoints** (void \*pabyX, int nXStride, void \*pabyY, int nYStride, void \*pabyZ=NULL, int nZStride=0) const  
*Returns all points of line string.*
- void **addSubLineString** (const **OGRLineString** \*, int nStartVertex=0, int nEndVertex=-1)  
*Add a segment of another linestring to this one.*
- void **reversePoints** (void)  
*Reverse point order.*
- virtual **OGRwkbGeometryType** **getGeometryType** () const  
*Fetch geometry type.*
- virtual const char \* **getGeometryName** () const  
*Fetch WKT name for geometry type.*
- virtual OGRErr **transform** (**OGRCoordinateTransformation** \*poCT)  
*Apply arbitrary coordinate transformation to geometry.*
- virtual void **flattenTo2D** ()  
*Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.*
- virtual void **segmentize** (double dfMaxLength)  
*Modify the geometry such it has no segment longer then the given distance.*
- virtual void **swapXY** ()  
*Swap x and y coordinates.*

### 12.64.1 Detailed Description

Concrete representation of a multi-vertex line.

### 12.64.2 Member Function Documentation

#### 12.64.2.1 void OGRLineString::addPoint ( OGRPoint \* poPoint )

Add a point to a line string.

The vertex count of the line string is increased by one, and assigned from the passed location value.

There is no SFCOM analog to this method.

## Parameters

<i>poPoint</i>	the point to assign to the new vertex.
----------------	--

References OGRGeometry::getCoordinateDimension(), OGRPoint::getX(), OGRPoint::getY(), OGRPoint::getZ(), and setPoint().

Referenced by OGRLinearRing::closeRings(), OGRBuildPolygonFromEdges(), and OGRLayer::SetSpatialFilterRect().

### 12.64.2.2 void OGRLineString::addPoint ( double x, double y, double z )

Add a point to a line string.

The vertex count of the line string is increased by one, and assigned from the passed location value.

There is no SFCOM analog to this method.

## Parameters

<i>x</i>	the X coordinate to assign to the new point.
<i>y</i>	the Y coordinate to assign to the new point.
<i>z</i>	the Z coordinate to assign to the new point (defaults to zero).

References setPoint().

### 12.64.2.3 void OGRLineString::addSubLineString ( const OGRLineString \* poOtherLine, int nStartVertex = 0, int nEndVertex = -1 )

Add a segment of another linestring to this one.

Adds the request range of vertices to the end of this line string in an efficient manner. If the nStartVertex is larger than the nEndVertex then the vertices will be reversed as they are copied.

## Parameters

<i>poOtherLine</i>	the other <b>OGRLineString</b> (p. ??).
<i>nStartVertex</i>	the first vertex to copy, defaults to 0 to start with the first vertex in the other linestring.
<i>nEndVertex</i>	the last vertex to copy, defaults to -1 indicating the last vertex of the other line string.

References getNumPoints(), and setNumPoints().

Referenced by OGRGeometryFactory::forceToLineString(), and OGRGeometryFactory::forceToMultiLineString().

### 12.64.2.4 OGRGeometry \* OGRLineString::clone ( ) const [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR\_G\_Clone()** (p. ??).

## Returns

a new object instance with the same geometry, and spatial reference system as the original.

Implements **OGRGeometry** (p. ??).

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

References OGRGeometry::assignSpatialReference(), OGRGeometry::getCoordinateDimension(), OGRGeometry::getSpatialReference(), OGRLineString::setCoordinateDimension(), and setPoints().

#### 12.64.2.5 void OGRLineString::empty ( ) [virtual]

Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.

This method relates to the SFCOM IGeometry::Empty() method.

This method is the same as the C function **OGR\_G\_Empty()** (p. ??).

Implements **OGRGeometry** (p. ??).

References setNumPoints().

Referenced by importFromWkt().

#### 12.64.2.6 void OGRLineString::EndPoint ( OGRPoint \* poPoint ) const [virtual]

Return the curve end point.

This method relates to the SF COM ICurve::get\_EndPoint() method.

##### Parameters

<i>poPoint</i>	the point to be assigned the end location.
----------------	--

Implements **OGRCurve** (p. ??).

References getPoint().

Referenced by OGRGeometryFactory::forceToLineString(), and Value().

#### 12.64.2.7 OGRBoolean OGRLineString::Equals ( OGRGeometry \* poOtherGeom ) const [virtual]

Returns TRUE if two geometries are equivalent.

This method is the same as the C function **OGR\_G\_Equals()** (p. ??).

##### Returns

TRUE if equivalent or FALSE otherwise.

Implements **OGRGeometry** (p. ??).

References OGRGeometry::getGeometryType(), getGeometryType(), getNumPoints(), getX(), getY(), and getZ().

Referenced by OGRPolygon::Equals().

#### 12.64.2.8 OGRErr OGRLineString::exportToWkb ( OGRwkbByteOrder eByteOrder, unsigned char \* pabyData ) const [virtual]

Convert a geometry into well known binary format.

This method relates to the SFCOM IWks::ExportToWKB() method.

This method is the same as the C function **OGR\_G\_ExportToWkb()** (p. ??).

##### Parameters

<i>eByteOrder</i>	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
<i>pabyData</i>	a buffer into which the binary representation is written. This buffer must be at least <b>OGRGeometry::WkbSize()</b> (p. ??) byte in size.

**Returns**

Currently OGRERR\_NONE is always returned.

Implements **OGRGeometry** (p. ??).

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

References OGRGeometry::getCoordinateDimension(), and getGeometryType().

**12.64.2.9 OGRErr OGRLineString::exportToWkt ( char \*\* ppszDstText ) const [virtual]**

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR\_G\_ExportToWkt()** (p. ??).

**Parameters**

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with OGRFree().
--------------------	--

**Returns**

Currently OGRERR\_NONE is always returned.

Implements **OGRGeometry** (p. ??).

References CPLDebug(), CPLStrdup(), OGRGeometry::getCoordinateDimension(), and getGeometryName().

Referenced by OGRPolygon::exportToWkt().

**12.64.2.10 void OGRLineString::flattenTo2D ( ) [virtual]**

Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.

This method is the same as the C function **OGR\_G\_FlattenTo2D()** (p. ??).

Implements **OGRGeometry** (p. ??).

**12.64.2.11 double OGRLineString::get\_Length ( ) const [virtual]**

Returns the length of the curve.

This method relates to the SFCOM ICurve::get\_Length() method.

**Returns**

the length of the curve, zero if the curve hasn't been initialized.

Implements **OGRCurve** (p. ??).

**12.64.2.12 int OGRLineString::getDimension ( ) const [virtual]**

Get the dimension of this object.

This method corresponds to the SFCOM IGeometry::GetDimension() method. It indicates the dimension of the object, but does not indicate the dimension of the underlying space (as indicated by **OGRGeometry::getCoordinateDimension()** (p. ??)).

This method is the same as the C function **OGR\_G\_GetDimension()** (p. ??).

**Returns**

0 for points, 1 for lines and 2 for surfaces.

Implements **OGRGeometry** (p. ??).

**12.64.2.13** `void OGRLineString::getEnvelope ( OGREnvelope * psEnvelope ) const` [virtual]

Computes and returns the bounding envelope for this geometry in the passed *psEnvelope* structure.

This method is the same as the C function **OGR\_G\_GetEnvelope()** (p. ??).

**Parameters**

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

Implements **OGRGeometry** (p. ??).

Referenced by `getEnvelope()`, and `OGRPolygon::getEnvelope()`.

**12.64.2.14** `void OGRLineString::getEnvelope ( OGREnvelope3D * psEnvelope ) const` [virtual]

Computes and returns the bounding envelope (3D) for this geometry in the passed *psEnvelope* structure.

This method is the same as the C function **OGR\_G\_GetEnvelope3D()** (p. ??).

**Parameters**

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

**Since**

OGR 1.9.0

Implements **OGRGeometry** (p. ??).

References `getEnvelope()`.

**12.64.2.15** `const char * OGRLineString::getGeometryName ( ) const` [virtual]

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR\_G\_GetGeometryName()** (p. ??).

**Returns**

name used for this geometry type in well known text format. The returned pointer is to a static internal string and should not be modified or freed.

Implements **OGRGeometry** (p. ??).

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

Referenced by `exportToWkt()`, and `importFromWkt()`.

**12.64.2.16** `OGRwkbGeometryType OGRLineString::getGeometryType ( ) const` [virtual]

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the `wkbFlatten()` macro to the return result.

This method is the same as the C function **OGR\_G\_GetGeometryType()** (p. ??).

**Returns**

the geometry type code.

Implements **OGRGeometry** (p. ??).

References OGRGeometry::getCoordinateDimension(), wkbLineString, and wkbLineString25D.

Referenced by Equals(), and exportToWkb().

### 12.64.2.17 int OGRLineString::getNumPoints ( ) const [inline]

Fetch vertex count.

Returns the number of vertices in the line string.

**Returns**

vertex count.

Referenced by addSubLineString(), OGRGeometry::dumpReadable(), Equals(), OGRGeometryFactory::forceToLineString(), OGRGeometryFactory::forceToMultiLineString(), OGR\_G\_GetPoint(), OGR\_G\_GetPointCount(), OGR\_G\_GetPoints(), OGR\_G\_GetX(), OGR\_G\_GetY(), OGR\_G\_GetZ(), OGRBuildPolygonFromEdges(), and OGRGeometryFactory::organizePolygons().

### 12.64.2.18 void OGRLineString::getPoint ( int *i*, OGRPoint \* *poPoint* ) const

Fetch a point in line string.

This method relates to the SFCOM ILineString::get\_Point() method.

**Parameters**

<i>i</i>	the vertex to fetch, from 0 to <b>getNumPoints()</b> (p. ??)-1.
<i>poPoint</i>	a point to initialize with the fetched point.

References OGRGeometry::getCoordinateDimension(), OGRPoint::setX(), OGRPoint::setY(), and OGRPoint::setZ().

Referenced by OGRLinearRing::closeRings(), EndPoint(), OGRGeometryFactory::organizePolygons(), and StartPoint().

### 12.64.2.19 void OGRLineString::getPoints ( OGRRawPoint \* *paoPointsOut*, double \* *padfZ* = NULL ) const

Returns all points of line string.

This method copies all points into user list. This list must be at least sizeof(OGRRawPoint) \* OGRGeometry::getNumPoints() byte in size. It also copies all Z coordinates.

There is no SFCOM analog to this method.

**Parameters**

<i>paoPointsOut</i>	a buffer into which the points is written.
<i>padfZ</i>	the Z values that go with the points (optional, may be NULL).

Referenced by getPoints(), and OGR\_G\_GetPoints().

### 12.64.2.20 void OGRLineString::getPoints ( void \* *pabyX*, int *nXStride*, void \* *pabyY*, int *nYStride*, void \* *pabyZ* = NULL, int *nZStride* = 0 ) const

Returns all points of line string.

This method copies all points into user arrays. The user provides the stride between 2 consecutives elements of the array.

On some CPU architectures, care must be taken so that the arrays are properly aligned.

There is no SFCOM analog to this method.

#### Parameters

<i>pabyX</i>	a buffer of at least (sizeof(double) * nXStride * nPointCount) bytes, may be NULL.
<i>nXStride</i>	the number of bytes between 2 elements of pabyX.
<i>pabyY</i>	a buffer of at least (sizeof(double) * nYStride * nPointCount) bytes, may be NULL.
<i>nYStride</i>	the number of bytes between 2 elements of pabyY.
<i>pabyZ</i>	a buffer of at last size (sizeof(double) * nZStride * nPointCount) bytes, may be NULL.
<i>nZStride</i>	the number of bytes between 2 elements of pabyZ.

#### Since

OGR 1.9.0

References `getPoints()`.

**12.64.2.21** `double OGRLineString::getX ( int iVertex ) const` `[inline]`

Get X at vertex.

Returns the X value at the indicated vertex. If *iVertex* is out of range a crash may occur, no internal range checking is performed.

#### Parameters

<i>iVertex</i>	the vertex to return, between 0 and <b>getNumPoints()</b> (p. ??)-1.
----------------	--

#### Returns

X value.

Referenced by `OGRLinearRing::closeRings()`, `Equals()`, `OGR_G_GetPoint()`, `OGR_G_GetX()`, and `OGRBuildPolygonFromEdges()`.

**12.64.2.22** `double OGRLineString::getY ( int iVertex ) const` `[inline]`

Get Y at vertex.

Returns the Y value at the indicated vertex. If *iVertex* is out of range a crash may occur, no internal range checking is performed.

#### Parameters

<i>iVertex</i>	the vertex to return, between 0 and <b>getNumPoints()</b> (p. ??)-1.
----------------	--

#### Returns

X value.

Referenced by `OGRLinearRing::closeRings()`, `Equals()`, `OGR_G_GetPoint()`, `OGR_G_GetY()`, and `OGRBuildPolygonFromEdges()`.

**12.64.2.23** `double OGRLineString::getZ ( int iVertex ) const`

Get Z at vertex.



Returns the Z (elevation) value at the indicated vertex. If no Z value is available, 0.0 is returned. If iVertex is out of range a crash may occur, no internal range checking is performed.

## Parameters

<i>iVertex</i>	the vertex to return, between 0 and <b>getNumPoints()</b> (p. ??)-1.
----------------	--

## Returns

Z value.

Referenced by **OGRLinearRing::closeRings()**, **Equals()**, **OGR\_G\_GetPoint()**, **OGR\_G\_GetZ()**, and **OGRBuildPolygonFromEdges()**.

**12.64.2.24 OGRErr OGRLinearString::importFromWkb ( unsigned char \* *pabyData*, int *nSize* = -1 ) [virtual]**

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the **SFCOM IWks::ImportFromWKB()** method.

This method is the same as the C function **OGR\_G\_ImportFromWkb()** (p. ??).

## Parameters

<i>pabyData</i>	the binary input data.
<i>nSize</i>	the size of <i>pabyData</i> in bytes, or zero if not known.

## Returns

**OGRERR\_NONE** if all goes well, otherwise any of **OGRERR\_NOT\_ENOUGH\_DATA**, **OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE**, or **OGRERR\_CORRUPT\_DATA** may be returned.

Implements **OGRGeometry** (p. ??).

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

References **CPLError()**, **setNumPoints()**, and **wkbLineString**.

**12.64.2.25 OGRErr OGRLinearString::importFromWkt ( char \*\* *ppszInput* ) [virtual]**

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the **SFCOM IWks::ImportFromWKT()** method.

This method is the same as the C function **OGR\_G\_ImportFromWkt()** (p. ??).

## Parameters

<i>ppszInput</i>	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.
------------------	---

## Returns

**OGRERR\_NONE** if all goes well, otherwise any of **OGRERR\_NOT\_ENOUGH\_DATA**, **OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE**, or **OGRERR\_CORRUPT\_DATA** may be returned.

Implements **OGRGeometry** (p. ??).

References **empty()**, and **getGeometryName()**.

**12.64.2.26** OGRBoolean OGRLineString::IsEmpty ( ) const [virtual]

Returns TRUE (non-zero) if the object has no points.

Normally this returns FALSE except between when an object is instantiated and points have been assigned.

This method relates to the SFCOM IGeometry::IsEmpty() method.

**Returns**

TRUE if object is empty, otherwise FALSE.

Implements **OGRGeometry** (p. ??).

**12.64.2.27** void OGRLineString::reversePoints ( void )

Reverse point order.

This method updates the points in this line string in place reversing the point ordering (first for last, etc).

Referenced by OGRGeometryFactory::forceToLineString().

**12.64.2.28** void OGRLineString::segmentize ( double *dfMaxLength* ) [virtual]

Modify the geometry such it has no segment longer then the given distance.

Interpolated points will have Z and M values (if needed) set to 0. Distance computation is performed in 2d only

This function is the same as the C function **OGR\_G\_Segmentize()** (p. ??)

**Parameters**

<i>dfMaxLength</i>	the maximum distance between 2 points after segmentization
--------------------	--

Reimplemented from **OGRGeometry** (p. ??).

References CPLError(), and OGRGeometry::getCoordinateDimension().

**12.64.2.29** void OGRLineString::setCoordinateDimension ( int *nNewDimension* ) [virtual]

Set the coordinate dimension.

This method sets the explicit coordinate dimension. Setting the coordinate dimension of a geometry to 2 should zero out any existing Z values. Setting the dimension of a geometry collection will not necessarily affect the children geometries.

**Parameters**

<i>nNewDimension</i>	New coordinate dimension value, either 2 or 3.
----------------------	--

Reimplemented from **OGRGeometry** (p. ??).

Referenced by clone(), and OGRPolygon::exportToWkt().

**12.64.2.30** void OGRLineString::setNumPoints ( int *nNewPointCount* )

Set number of points in geometry.

This method primary exists to preset the number of points in a linestring geometry before **setPoint()** (p. ??) is used to assign them to avoid reallocating the array larger with each call to **addPoint()** (p. ??).

This method has no SFCOM analog.

## Parameters

<i>nNewPointCount</i>	the new number of points for geometry.
-----------------------	--

References CPLError(), and OGRGeometry::getCoordinateDimension().

Referenced by addSubLineString(), empty(), importFromWkb(), setPoint(), and setPoints().

#### 12.64.2.31 void OGRLineString::setPoint ( int *iPoint*, OGRPoint \* *poPoint* )

Set the location of a vertex in line string.

If *iPoint* is larger than the number of necessary the number of existing points in the line string, the point count will be increased to accomodate the request.

There is no SFCOM analog to this method.

## Parameters

<i>iPoint</i>	the index of the vertex to assign (zero based).
<i>poPoint</i>	the value to assign to the vertex.

References OGRGeometry::getCoordinateDimension(), OGRPoint::getX(), OGRPoint::getY(), and OGRPoint::getZ().

Referenced by addPoint(), and OGRGeometryFactory::approximateArcAngles().

#### 12.64.2.32 void OGRLineString::setPoint ( int *iPoint*, double *xIn*, double *yIn*, double *zIn* )

Set the location of a vertex in line string.

If *iPoint* is larger than the number of necessary the number of existing points in the line string, the point count will be increased to accomodate the request.

There is no SFCOM analog to this method.

## Parameters

<i>iPoint</i>	the index of the vertex to assign (zero based).
<i>xIn</i>	input X coordinate to assign.
<i>yIn</i>	input Y coordinate to assign.
<i>zIn</i>	input Z coordinate to assign (defaults to zero).

References OGRGeometry::getCoordinateDimension(), and setNumPoints().

#### 12.64.2.33 void OGRLineString::setPoints ( int *nPointsIn*, OGRRawPoint \* *paoPointsIn*, double \* *padfZ* = NULL )

Assign all points in a line string.

This method clears any existing points assigned to this line string, and assigns a whole new set. It is the most efficient way of assigning the value of a line string.

There is no SFCOM analog to this method.

## Parameters

<i>nPointsIn</i>	number of points being passed in <i>paoPointsIn</i>
<i>paoPointsIn</i>	list of points being assigned.
<i>padfZ</i>	the Z values that go with the points (optional, may be NULL).

References OGRGeometry::getCoordinateDimension(), and setNumPoints().

Referenced by clone(), OGRLinearRing::clone(), OGRPolygon::importFromWkt(), OGRMultiPolygon::importFromWkt(), OGRMultiLineString::importFromWkt(), and transform().

**12.64.2.34** void OGRLineString::setPoints ( int *nPointsIn*, double \* *padfX*, double \* *padfY*, double \* *padfZ* = NULL )

Assign all points in a line string.

This method clear any existing points assigned to this line string, and assigns a whole new set.

There is no SFCOM analog to this method.

#### Parameters

<i>nPointsIn</i>	number of points being passed in padfX and padfY.
<i>padfX</i>	list of X coordinates of points being assigned.
<i>padfY</i>	list of Y coordinates of points being assigned.
<i>padfZ</i>	list of Z coordinates of points being assigned (defaults to NULL for 2D objects).

References setNumPoints().

**12.64.2.35** void OGRLineString::StartPoint ( OGRPoint \* *poPoint* ) const [virtual]

Return the curve start point.

This method relates to the SF COM ICurve::get\_StartPoint() method.

#### Parameters

<i>poPoint</i>	the point to be assigned the start location.
----------------	--

Implements **OGRCurve** (p. ??).

References getPoint().

Referenced by OGRGeometryFactory::forceToLineString(), and Value().

**12.64.2.36** void OGRLineString::swapXY ( ) [virtual]

Swap x and y coordinates.

Since

OGR 1.8.0

Reimplemented from **OGRGeometry** (p. ??).

**12.64.2.37** OGRErr OGRLineString::transform ( OGRCoordinateTransformation \* *poCT* ) [virtual]

Apply arbitrary coordinate transformation to geometry.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

Note that this method does not require that the geometry already have a spatial reference system. It will be assumed that they can be treated as having the source spatial reference system of the **OGRCoordinateTransformation** (p. ??) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGR-SpatialReference** (p. ??) of the **OGRCoordinateTransformation** (p. ??) will be assigned to the geometry.

This method is the same as the C function **OGR\_G\_Transform()** (p. ??).

## Parameters

<i>poCT</i>	the transformation to apply.
-------------	------------------------------

## Returns

OGRERR\_NONE on success or an error code.

Implements **OGRGeometry** (p. ??).

References OGRGeometry::assignSpatialReference(), CPLError(), CPLGetConfigOption(), CSLTestBoolean(), OGRCoordinateTransformation::GetTargetCS(), setPoints(), and OGRCoordinateTransformation::TransformEx().

Referenced by OGRPolygon::transform().

**12.64.2.38** void OGRLineString::Value ( double *dfDistance*, OGRPoint \* *poPoint* ) const [virtual]

Fetch point at given distance along curve.

This method relates to the SF COM ICurve::get\_Value() method.

## Parameters

<i>dfDistance</i>	distance along the curve at which to sample position. This distance should be between zero and <b>get_Length()</b> (p. ??) for this curve.
<i>poPoint</i>	the point to be assigned the curve position.

Implements **OGRCurve** (p. ??).

References EndPoint(), OGRGeometry::getCoordinateDimension(), OGRPoint::setX(), OGRPoint::setY(), OGRPoint::setZ(), and StartPoint().

**12.64.2.39** int OGRLineString::WkbSize ( ) const [virtual]

Returns size of related binary representation.

This method returns the exact number of bytes required to hold the well known binary representation of this geometry object. Its computation may be slightly expensive for complex geometries.

This method relates to the SFCOM IWks::WkbSize() method.

This method is the same as the C function **OGR\_G\_WkbSize()** (p. ??).

## Returns

size of binary representation in bytes.

Implements **OGRGeometry** (p. ??).

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

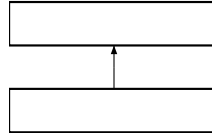
References OGRGeometry::getCoordinateDimension().

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

- **ogr\_geometry.h**
- **ogrlinestring.cpp**

## 12.65 OGRMIDatIndex Class Reference

Inheritance diagram for OGRMIDatIndex:

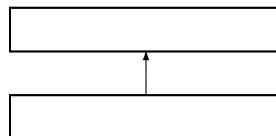


The documentation for this class was generated from the following file:

- ogr\_miattrind.cpp

## 12.66 OGRMILayerAttrIndex Class Reference

Inheritance diagram for OGRMILayerAttrIndex:



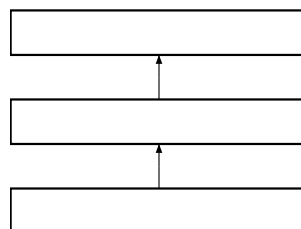
The documentation for this class was generated from the following file:

- ogr\_miattrind.cpp

## 12.67 OGRMultiLineString Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRMultiLineString:



### Public Member Functions

- virtual const char \* **getGeometryName** () const  
*Fetch WKT name for geometry type.*
- virtual **OGRwkbGeometryType** **getGeometryType** () const  
*Fetch geometry type.*
- virtual **OGRGeometry** \* **clone** () const  
*Make a copy of this object.*
- virtual OGRErr **importFromWkt** (char \*\*) *Assign geometry from well known text data.*
- virtual OGRErr **exportToWkt** (char \*\*) const



*Convert a geometry into well known text format.*

- virtual int **getDimension** () const

*Get the dimension of this object.*

- virtual OGRErr **addGeometryDirectly** (OGRGeometry \*)

*Add a geometry directly to the container.*

### 12.67.1 Detailed Description

A collection of OGRLineStrings.

### 12.67.2 Member Function Documentation

#### 12.67.2.1 OGRErr OGRMultiLineString::addGeometryDirectly ( OGRGeometry \* poNewGeom ) [virtual]

Add a geometry directly to the container.

Some subclasses of **OGRGeometryCollection** (p. ??) restrict the types of geometry that can be added, and may return an error. Ownership of the passed geometry is taken by the container rather than cloning as **addGeometry()** (p. ??) does.

This method is the same as the C function **OGR\_G\_AddGeometryDirectly()** (p. ??).

There is no SFCOM analog to this method.

#### Parameters

<i>poNewGeom</i>	geometry to add to the container.
------------------	-----------------------------------

#### Returns

OGRERR\_NONE if successful, or OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE if the geometry type is illegal for the type of geometry container.

Reimplemented from **OGRGeometryCollection** (p. ??).

References OGRGeometryCollection::addGeometryDirectly(), OGRGeometry::getGeometryType(), wkbLineString, and wkbLineString25D.

Referenced by OGRGeometryFactory::forceToMultiLineString(), and importFromWkt().

#### 12.67.2.2 OGRGeometry \* OGRMultiLineString::clone ( ) const [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR\_G\_Clone()** (p. ??).

#### Returns

a new object instance with the same geometry, and spatial reference system as the original.

Reimplemented from **OGRGeometryCollection** (p. ??).

References OGRGeometryCollection::addGeometry(), OGRGeometry::assignSpatialReference(), OGRGeometryCollection::getGeometryRef(), OGRGeometryCollection::getNumGeometries(), and OGRGeometry::getSpatialReference().

### 12.67.2.3 OGRErr OGRMultiLineString::exportToWkt ( char \*\* *ppszDstText* ) const [virtual]

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR\_G\_ExportToWkt()** (p. ??).

#### Parameters

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with OGRFree().
--------------------	--

#### Returns

Currently OGRERR\_NONE is always returned.

Reimplemented from **OGRGeometryCollection** (p. ??).

References CPLCalloc(), CPLDebug(), CPLStrdup(), OGRGeometry::exportToWkt(), OGRGeometryCollection::getGeometryRef(), and OGRGeometryCollection::getNumGeometries().

### 12.67.2.4 int OGRMultiLineString::getDimension ( ) const [virtual]

Get the dimension of this object.

This method corresponds to the SFCOM IGeometry::GetDimension() method. It indicates the dimension of the object, but does not indicate the dimension of the underlying space (as indicated by **OGRGeometry::getCoordinateDimension()** (p. ??)).

This method is the same as the C function **OGR\_G\_GetDimension()** (p. ??).

#### Returns

0 for points, 1 for lines and 2 for surfaces.

Reimplemented from **OGRGeometryCollection** (p. ??).

### 12.67.2.5 const char \* OGRMultiLineString::getGeometryName ( ) const [virtual]

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR\_G\_GetGeometryName()** (p. ??).

#### Returns

name used for this geometry type in well known text format. The returned pointer is to a static internal string and should not be modified or freed.

Reimplemented from **OGRGeometryCollection** (p. ??).

Referenced by importFromWkt().

### 12.67.2.6 OGRwkbGeometryType OGRMultiLineString::getGeometryType ( ) const [virtual]

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the wkbFlatten() macro to the return result.

This method is the same as the C function **OGR\_G\_GetGeometryType()** (p. ??).

**Returns**

the geometry type code.

Reimplemented from **OGRGeometryCollection** (p. ??).

References OGRGeometry::getCoordinateDimension(), wkbMultiLineString, and wkbMultiLineString25D.

**12.67.2.7 OGRErr OGRMultiLineString::importFromWkt ( char \*\* ppszInput ) [virtual]**

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR\_G\_ImportFromWkt()** (p. ??).

**Parameters**

<i>ppszInput</i>	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.
------------------	---

**Returns**

OGRErr\_NONE if all goes well, otherwise any of OGRErr\_NOT\_ENOUGH\_DATA, OGRErr\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRErr\_CORRUPT\_DATA may be returned.

Reimplemented from **OGRGeometryCollection** (p. ??).

References addGeometryDirectly(), OGRGeometryCollection::empty(), getGeometryName(), and OGRLineString::setPoints().

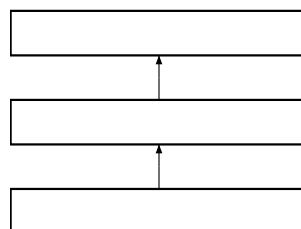
The documentation for this class was generated from the following files:

- **ogr\_geometry.h**
- **ogrmultilinestring.cpp**

**12.68 OGRMultiPoint Class Reference**

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRMultiPoint:

**Public Member Functions**

- virtual const char \* **getGeometryName** () const  
*Fetch WKT name for geometry type.*
- virtual **OGRwkbGeometryType** **getGeometryType** () const

*Fetch geometry type.*

- virtual **OGRGeometry** \* **clone** () const

*Make a copy of this object.*

- virtual OGRErr **importFromWkt** (char \*\*)

*Assign geometry from well known text data.*

- virtual OGRErr **exportToWkt** (char \*\*) const

*Convert a geometry into well known text format.*

- virtual int **getDimension** () const

*Get the dimension of this object.*

- virtual OGRErr **addGeometryDirectly** (**OGRGeometry** \*)

*Add a geometry directly to the container.*

### 12.68.1 Detailed Description

A collection of OGRPoints.

### 12.68.2 Member Function Documentation

#### 12.68.2.1 OGRErr OGRMultiPoint::addGeometryDirectly ( **OGRGeometry** \* *poNewGeom* ) [virtual]

Add a geometry directly to the container.

Some subclasses of **OGRGeometryCollection** (p. ??) restrict the types of geometry that can be added, and may return an error. Ownership of the passed geometry is taken by the container rather than cloning as **addGeometry()** (p. ??) does.

This method is the same as the C function **OGR\_G\_AddGeometryDirectly()** (p. ??).

There is no SFCOM analog to this method.

#### Parameters

<i>poNewGeom</i>	geometry to add to the container.
------------------	-----------------------------------

#### Returns

OGRErr\_NONE if successful, or OGRErr\_UNSUPPORTED\_GEOMETRY\_TYPE if the geometry type is illegal for the type of geometry container.

Reimplemented from **OGRGeometryCollection** (p. ??).

References OGRGeometryCollection::addGeometryDirectly(), OGRGeometry::getGeometryType(), wkbPoint, and wkbPoint25D.

Referenced by OGRGeometryFactory::forceToMultiPoint(), and importFromWkt().

#### 12.68.2.2 **OGRGeometry** \* OGRMultiPoint::clone ( ) const [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR\_G\_Clone()** (p. ??).

**Returns**

a new object instance with the same geometry, and spatial reference system as the original.

Reimplemented from **OGRGeometryCollection** (p. ??).

References OGRGeometryCollection::addGeometry(), OGRGeometry::assignSpatialReference(), OGRGeometryCollection::getGeometryRef(), OGRGeometryCollection::getNumGeometries(), and OGRGeometry::getSpatialReference().

**12.68.2.3 OGRErr OGRMultiPoint::exportToWkt ( char \*\* ppszDstText ) const [virtual]**

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR\_G\_ExportToWkt()** (p. ??).

**Parameters**

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with OGRFree().
--------------------	--

**Returns**

Currently OGRErr\_NONE is always returned.

Reimplemented from **OGRGeometryCollection** (p. ??).

References CPLDebug(), CPLRealloc(), CPLStrdup(), OGRGeometry::getCoordinateDimension(), getGeometryName(), OGRGeometryCollection::getGeometryRef(), OGRGeometryCollection::getNumGeometries(), OGRPoint::getX(), OGRPoint::getY(), OGRPoint::getZ(), OGRPoint::IsEmpty(), and OGRGeometryCollection::IsEmpty().

**12.68.2.4 int OGRMultiPoint::getDimension ( ) const [virtual]**

Get the dimension of this object.

This method corresponds to the SFCOM IGeometry::GetDimension() method. It indicates the dimension of the object, but does not indicate the dimension of the underlying space (as indicated by **OGRGeometry::getCoordinateDimension()** (p. ??)).

This method is the same as the C function **OGR\_G\_GetDimension()** (p. ??).

**Returns**

0 for points, 1 for lines and 2 for surfaces.

Reimplemented from **OGRGeometryCollection** (p. ??).

**12.68.2.5 const char \* OGRMultiPoint::getGeometryName ( ) const [virtual]**

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR\_G\_GetGeometryName()** (p. ??).

**Returns**

name used for this geometry type in well known text format. The returned pointer is to a static internal string and should not be modified or freed.

Reimplemented from **OGRGeometryCollection** (p. ??).

Referenced by exportToWkt(), and importFromWkt().

### 12.68.2.6 OGRwkbGeometryType OGRMultiPoint::getGeometryType ( ) const [virtual]

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the `wkbFlatten()` macro to the return result.

This method is the same as the C function `OGR_G_GetGeometryType()` (p. ??).

#### Returns

the geometry type code.

Reimplemented from `OGRGeometryCollection` (p. ??).

References `OGRGeometry::getCoordinateDimension()`, `wkbMultiPoint`, and `wkbMultiPoint25D`.

### 12.68.2.7 OGRErr OGRMultiPoint::importFromWkt ( char \*\* ppszInput ) [virtual]

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the `OGRGeometryFactory` (p. ??) class, but not normally called by application code.

This method relates to the `SFCOM IWks::ImportFromWKT()` method.

This method is the same as the C function `OGR_G_ImportFromWkt()` (p. ??).

#### Parameters

<i>ppszInput</i>	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.
------------------	---

#### Returns

`OGRErr_NONE` if all goes well, otherwise any of `OGRErr_NOT_ENOUGH_DATA`, `OGRErr_UNSUPPORTED_GEOMETRY_TYPE`, or `OGRErr_CORRUPT_DATA` may be returned.

Reimplemented from `OGRGeometryCollection` (p. ??).

References `addGeometryDirectly()`, `OGRGeometryCollection::empty()`, and `getGeometryName()`.

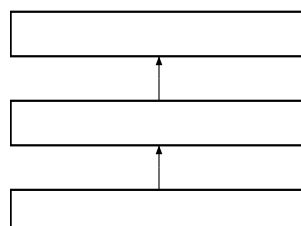
The documentation for this class was generated from the following files:

- `ogr_geometry.h`
- `ogrmultipoint.cpp`

## 12.69 OGRMultiPolygon Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for `OGRMultiPolygon`:



## Public Member Functions

- virtual const char \* **getGeometryName** () const  
*Fetch WKT name for geometry type.*
- virtual **OGRwkbGeometryType** **getGeometryType** () const  
*Fetch geometry type.*
- virtual **OGRGeometry** \* **clone** () const  
*Make a copy of this object.*
- virtual OGRErr **importFromWkt** (char \*\*)   
*Assign geometry from well known text data.*
- virtual OGRErr **exportToWkt** (char \*\*) const  
*Convert a geometry into well known text format.*
- virtual int **getDimension** () const  
*Get the dimension of this object.*
- virtual OGRErr **addGeometryDirectly** (**OGRGeometry** \*)  
*Add a geometry directly to the container.*
- virtual double **get\_Area** () const

### 12.69.1 Detailed Description

A collection of non-overlapping OGRPolygons.

Note that the IMultiSurface class hasn't been modelled, nor have any of it's methods.

### 12.69.2 Member Function Documentation

#### 12.69.2.1 OGRErr OGRMultiPolygon::addGeometryDirectly ( OGRGeometry \* poNewGeom ) [virtual]

Add a geometry directly to the container.

Some subclasses of **OGRGeometryCollection** (p. ??) restrict the types of geometry that can be added, and may return an error. Ownership of the passed geometry is taken by the container rather than cloning as **addGeometry()** (p. ??) does.

This method is the same as the C function **OGR\_G\_AddGeometryDirectly()** (p. ??).

There is no SFCOM analog to this method.

#### Parameters

<i>poNewGeom</i>	geometry to add to the container.
------------------	-----------------------------------

#### Returns

OGRErr\_NONE if successful, or OGRErr\_UNSUPPORTED\_GEOMETRY\_TYPE if the geometry type is illegal for the type of geometry container.

Reimplemented from **OGRGeometryCollection** (p. ??).

References **OGRGeometryCollection::addGeometryDirectly()**, **OGRGeometry::getGeometryType()**, **wkbPolygon**, and **wkbPolygon25D**.

Referenced by **OGRGeometryFactory::forceToMultiPolygon()**, and **importFromWkt()**.

### 12.69.2.2 **OGRGeometry \* OGRMultiPolygon::clone ( ) const** [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR\_G\_Clone()** (p. ??).

#### Returns

a new object instance with the same geometry, and spatial reference system as the original.

Reimplemented from **OGRGeometryCollection** (p. ??).

References OGRGeometryCollection::addGeometry(), OGRGeometry::assignSpatialReference(), OGRGeometryCollection::getGeometryRef(), OGRGeometryCollection::getNumGeometries(), and OGRGeometry::getSpatialReference().

### 12.69.2.3 **OGRERR OGRMultiPolygon::exportToWkt ( char \*\* ppszDstText ) const** [virtual]

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR\_G\_ExportToWkt()** (p. ??).

#### Parameters

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with OGRFree().
--------------------	--

#### Returns

Currently OGRERR\_NONE is always returned.

Reimplemented from **OGRGeometryCollection** (p. ??).

References CPLCalloc(), CPLDebug(), CPLStrdup(), OGRGeometry::exportToWkt(), OGRGeometryCollection::getGeometryRef(), and OGRGeometryCollection::getNumGeometries().

### 12.69.2.4 **double OGRMultiPolygon::get\_Area ( ) const** [virtual]

Compute area of multipolygon.

The area is computed as the sum of the areas of all polygon members in this collection.

#### Returns

computed area.

Reimplemented from **OGRGeometryCollection** (p. ??).

References OGRPolygon::get\_Area(), OGRGeometryCollection::getGeometryRef(), and OGRGeometryCollection::getNumGeometries().

### 12.69.2.5 **int OGRMultiPolygon::getDimension ( ) const** [virtual]

Get the dimension of this object.

This method corresponds to the SFCOM IGeometry::GetDimension() method. It indicates the dimension of the object, but does not indicate the dimension of the underlying space (as indicated by **OGRGeometry::getCoordinateDimension()** (p. ??)).

This method is the same as the C function **OGR\_G\_GetDimension()** (p. ??).



**Returns**

0 for points, 1 for lines and 2 for surfaces.

Reimplemented from **OGRGeometryCollection** (p. ??).

**12.69.2.6 const char \* OGRMultiPolygon::getGeometryName ( ) const [virtual]**

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR\_G\_GetGeometryName()** (p. ??).

**Returns**

name used for this geometry type in well known text format. The returned pointer is to a static internal string and should not be modified or freed.

Reimplemented from **OGRGeometryCollection** (p. ??).

Referenced by importFromWkt().

**12.69.2.7 OGRwkbGeometryType OGRMultiPolygon::getGeometryType ( ) const [virtual]**

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the wkbFlatten() macro to the return result.

This method is the same as the C function **OGR\_G\_GetGeometryType()** (p. ??).

**Returns**

the geometry type code.

Reimplemented from **OGRGeometryCollection** (p. ??).

References OGRGeometry::getCoordinateDimension(), wkbMultiPolygon, and wkbMultiPolygon25D.

**12.69.2.8 OGRErr OGRMultiPolygon::importFromWkt ( char \*\* ppszInput ) [virtual]**

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR\_G\_ImportFromWkt()** (p. ??).

**Parameters**

<i>ppszInput</i>	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.
------------------	---

**Returns**

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

Reimplemented from **OGRGeometryCollection** (p. ??).

References `addGeometryDirectly()`, `OGRPolygon::addRingDirectly()`, `OGRGeometryCollection::empty()`, `getGeometryName()`, and `OGRLineString::setPoints()`.

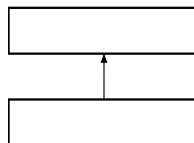
The documentation for this class was generated from the following files:

- `ogr_geometry.h`
- `ogrmultipolygon.cpp`

## 12.70 OGRPoint Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRPoint:



### Public Member Functions

- **OGRPoint** ()  
*Create a (0,0) point.*
- virtual int **WkbSize** () const  
*Returns size of related binary representation.*
- virtual OGRErr **importFromWkb** (unsigned char \*, int=-1)  
*Assign geometry from well known binary data.*
- virtual OGRErr **exportToWkb** (OGRwkbByteOrder, unsigned char \*) const  
*Convert a geometry into well known binary format.*
- virtual OGRErr **importFromWkt** (char \*\*) *Assign geometry from well known text data.*
- virtual OGRErr **exportToWkt** (char \*\*pszDstText) const  
*Convert a geometry into well known text format.*
- virtual int **getDimension** () const  
*Get the dimension of this object.*
- virtual **OGRGeometry** \* **clone** () const  
*Make a copy of this object.*
- virtual void **empty** ()  
*Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.*
- virtual void **getEnvelope** (**OGREnvelope** \*psEnvelope) const  
*Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.*
- virtual void **getEnvelope** (**OGREnvelope3D** \*psEnvelope) const  
*Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.*
- virtual OGRBoolean **IsEmpty** () const  
*Returns TRUE (non-zero) if the object has no points.*
- double **getX** () const  
*Fetch X coordinate.*
- double **getY** () const  
*Fetch Y coordinate.*
- double **getZ** () const

- Fetch Z coordinate.*
- virtual void **setCoordinateDimension** (int nDimension)
- Set the coordinate dimension.*
- void **setX** (double xIn)
- Assign point X coordinate.*
- void **setY** (double yIn)
- Assign point Y coordinate.*
- void **setZ** (double zIn)
- Assign point Z coordinate. Calling this method will force the geometry coordinate dimension to 3D (wkbPoint|wkbZ).*
- virtual OGRBoolean **Equals** (OGRGeometry \*) const
- Returns TRUE if two geometries are equivalent.*
- virtual const char \* **getGeometryName** () const
- Fetch WKT name for geometry type.*
- virtual OGRwkbGeometryType **getGeometryType** () const
- Fetch geometry type.*
- virtual OGRErr **transform** (OGRCoordinateTransformation \*poCT)
- Apply arbitrary coordinate transformation to geometry.*
- virtual void **flattenTo2D** ()
- Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.*
- virtual void **swapXY** ()
- Swap x and y coordinates.*

### 12.70.1 Detailed Description

Point class.

Implements SFCOM IPoint methods.

### 12.70.2 Member Function Documentation

#### 12.70.2.1 OGRGeometry \* OGRPoint::clone ( ) const [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR\_G\_Clone()** (p. ??).

#### Returns

a new object instance with the same geometry, and spatial reference system as the original.

Implements **OGRGeometry** (p. ??).

References **OGRGeometry::assignSpatialReference()**, **OGRGeometry::getSpatialReference()**, **OGRPoint()**, and **setCoordinateDimension()**.

#### 12.70.2.2 void OGRPoint::empty ( ) [virtual]

Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.

This method relates to the SFCOM IGeometry::Empty() method.

This method is the same as the C function **OGR\_G\_Empty()** (p. ??).

Implements **OGRGeometry** (p. ??).

Referenced by **OGRGeometry::Centroid()**, **importFromWkt()**, **OGRPoint()**, and **OGRPolygon::PointOnSurface()**.

### 12.70.2.3 OGRBoolean OGRPoint::Equals ( OGRGeometry \* *poOtherGeom* ) const [virtual]

Returns TRUE if two geometries are equivalent.

This method is the same as the C function **OGR\_G\_Equals()** (p. ??).

#### Returns

TRUE if equivalent or FALSE otherwise.

Implements **OGRGeometry** (p. ??).

References OGRGeometry::getGeometryType(), getGeometryType(), getX(), getY(), and getZ().

Referenced by OGRGeometryFactory::forceToLineString().

### 12.70.2.4 OGRErr OGRPoint::exportToWkb ( OGRwkbByteOrder *eByteOrder*, unsigned char \* *pabyData* ) const [virtual]

Convert a geometry into well known binary format.

This method relates to the SFCOM IWks::ExportToWKB() method.

This method is the same as the C function **OGR\_G\_ExportToWkb()** (p. ??).

#### Parameters

<i>eByteOrder</i>	One of wkbXDR or wkbNDR indicating MSB or LSB byte order respectively.
<i>pabyData</i>	a buffer into which the binary representation is written. This buffer must be at least <b>OGR-Geometry::WkbSize()</b> (p. ??) byte in size.

#### Returns

Currently OGRERR\_NONE is always returned.

Implements **OGRGeometry** (p. ??).

References getGeometryType().

### 12.70.2.5 OGRErr OGRPoint::exportToWkt ( char \*\* *ppszDstText* ) const [virtual]

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR\_G\_ExportToWkt()** (p. ??).

#### Parameters

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with OGRFree().
--------------------	--

#### Returns

Currently OGRERR\_NONE is always returned.

Implements **OGRGeometry** (p. ??).

References CPLStrdup().

### 12.70.2.6 void OGRPoint::flattenTo2D ( ) [virtual]

Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.

This method is the same as the C function **OGR\_G\_FlattenTo2D()** (p. ??).

Implements **OGRGeometry** (p. ??).

#### 12.70.2.7 int OGRPoint::getDimension ( ) const [virtual]

Get the dimension of this object.

This method corresponds to the SFCOM IGeometry::GetDimension() method. It indicates the dimension of the object, but does not indicate the dimension of the underlying space (as indicated by **OGRGeometry::getCoordinateDimension()** (p. ??)).

This method is the same as the C function **OGR\_G\_GetDimension()** (p. ??).

#### Returns

0 for points, 1 for lines and 2 for surfaces.

Implements **OGRGeometry** (p. ??).

#### 12.70.2.8 void OGRPoint::getEnvelope ( OGREnvelope \* *psEnvelope* ) const [virtual]

Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR\_G\_GetEnvelope()** (p. ??).

#### Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

Implements **OGRGeometry** (p. ??).

References getX(), and getY().

#### 12.70.2.9 void OGRPoint::getEnvelope ( OGREnvelope3D \* *psEnvelope* ) const [virtual]

Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR\_G\_GetEnvelope3D()** (p. ??).

#### Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

#### Since

OGR 1.9.0

Implements **OGRGeometry** (p. ??).

References getX(), getY(), and getZ().

#### 12.70.2.10 const char \* OGRPoint::getGeometryName ( ) const [virtual]

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR\_G\_GetGeometryName()** (p. ??).

**Returns**

name used for this geometry type in well known text format. The returned pointer is to a static internal string and should not be modified or freed.

Implements **OGRGeometry** (p. ??).

#### 12.70.2.11 **OGRwkbGeometryType OGRPoint::getGeometryType ( ) const** `[virtual]`

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the `wkbFlatten()` macro to the return result.

This method is the same as the C function **OGR\_G\_GetGeometryType()** (p. ??).

**Returns**

the geometry type code.

Implements **OGRGeometry** (p. ??).

References `wkbPoint`, and `wkbPoint25D`.

Referenced by `Equals()`, `exportToWkb()`, and `OGR_G_Centroid()`.

#### 12.70.2.12 **double OGRPoint::getX ( ) const** `[inline]`

Fetch X coordinate.

Relates to the SFCOM `IPoint::get_X()` method.

**Returns**

the X coordinate of this point.

Referenced by `OGRLineString::addPoint()`, `OGRGeometry::Centroid()`, `Equals()`, `OGRMultiPoint::exportToWkt()`, `OGRCurve::get_IsClosed()`, `getEnvelope()`, `OGRPolygon::PointOnSurface()`, and `OGRLineString::setPoint()`.

#### 12.70.2.13 **double OGRPoint::getY ( ) const** `[inline]`

Fetch Y coordinate.

Relates to the SFCOM `IPoint::get_Y()` method.

**Returns**

the Y coordinate of this point.

Referenced by `OGRLineString::addPoint()`, `OGRGeometry::Centroid()`, `Equals()`, `OGRMultiPoint::exportToWkt()`, `OGRCurve::get_IsClosed()`, `getEnvelope()`, `OGRPolygon::PointOnSurface()`, and `OGRLineString::setPoint()`.

#### 12.70.2.14 **double OGRPoint::getZ ( ) const** `[inline]`

Fetch Z coordinate.

Relates to the SFCOM `IPoint::get_Z()` method.

**Returns**

the Z coordinate of this point, or zero if it is a 2D point.

Referenced by `OGRLineString::addPoint()`, `Equals()`, `OGRMultiPoint::exportToWkt()`, `getEnvelope()`, and `OGRLineString::setPoint()`.

**12.70.2.15** OGRErr OGRPoint::importFromWkb ( unsigned char \* *pabyData*, int *nSize* = -1 ) [virtual]

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKB() method.

This method is the same as the C function **OGR\_G\_ImportFromWkb()** (p. ??).

**Parameters**

<i>pabyData</i>	the binary input data.
<i>nSize</i>	the size of pabyData in bytes, or zero if not known.

**Returns**

OGRErr\_NONE if all goes well, otherwise any of OGRErr\_NOT\_ENOUGH\_DATA, OGRErr\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRErr\_CORRUPT\_DATA may be returned.

Implements **OGRGeometry** (p. ??).

References wkbPoint.

**12.70.2.16** OGRErr OGRPoint::importFromWkt ( char \*\* *ppszInput* ) [virtual]

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR\_G\_ImportFromWkt()** (p. ??).

**Parameters**

<i>ppszInput</i>	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.
------------------	---

**Returns**

OGRErr\_NONE if all goes well, otherwise any of OGRErr\_NOT\_ENOUGH\_DATA, OGRErr\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRErr\_CORRUPT\_DATA may be returned.

Implements **OGRGeometry** (p. ??).

References empty().

**12.70.2.17** OGRBoolean OGRPoint::isEmpty ( ) const [virtual]

Returns TRUE (non-zero) if the object has no points.

Normally this returns FALSE except between when an object is instantiated and points have been assigned.

This method relates to the SFCOM IGeometry::isEmpty() method.

**Returns**

TRUE if object is empty, otherwise FALSE.

Implements **OGRGeometry** (p. ??).

Referenced by OGRGeometry::Centroid(), OGRMultiPoint::exportToWkt(), and OGRPolygon::PointOnSurface().

**12.70.2.18** void OGRPoint::setCoordinateDimension ( int *nNewDimension* ) [virtual]

Set the coordinate dimension.

This method sets the explicit coordinate dimension. Setting the coordinate dimension of a geometry to 2 should zero out any existing Z values. Setting the dimension of a geometry collection will not necessarily affect the children geometries.

#### Parameters

<i>nNewDimension</i>	New coordinate dimension value, either 2 or 3.
----------------------	--

Reimplemented from **OGRGeometry** (p. ??).

Referenced by clone().

**12.70.2.19** void OGRPoint::setX ( double *xIn* ) [inline]

Assign point X coordinate.

There is no corresponding SFCOM method.

Referenced by OGRGeometry::Centroid(), OGRLineString::getPoint(), OGRPolygon::PointOnSurface(), and OGRLineString::Value().

**12.70.2.20** void OGRPoint::setY ( double *yIn* ) [inline]

Assign point Y coordinate.

There is no corresponding SFCOM method.

Referenced by OGRGeometry::Centroid(), OGRLineString::getPoint(), OGRPolygon::PointOnSurface(), and OGRLineString::Value().

**12.70.2.21** void OGRPoint::setZ ( double *zIn* ) [inline]

Assign point Z coordinate. Calling this method will force the geometry coordinate dimension to 3D (wkbPoint|wkbZ).

There is no corresponding SFCOM method.

Referenced by OGRLineString::getPoint(), and OGRLineString::Value().

**12.70.2.22** void OGRPoint::swapXY ( ) [virtual]

Swap x and y coordinates.

#### Since

OGR 1.8.0

Reimplemented from **OGRGeometry** (p. ??).

**12.70.2.23** OGRErr OGRPoint::transform ( OGRCoordinateTransformation \* *poCT* ) [virtual]

Apply arbitrary coordinate transformation to geometry.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.



Note that this method does not require that the geometry already have a spatial reference system. It will be assumed that they can be treated as having the source spatial reference system of the **OGRCoordinateTransformation** (p. ??) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGRSpatialReference** (p. ??) of the **OGRCoordinateTransformation** (p. ??) will be assigned to the geometry.

This method is the same as the C function **OGR\_G\_Transform()** (p. ??).

#### Parameters

<i>poCT</i>	the transformation to apply.
-------------	------------------------------

#### Returns

OGRERR\_NONE on success or an error code.

Implements **OGRGeometry** (p. ??).

References **OGRGeometry::assignSpatialReference()**, **OGRCoordinateTransformation::GetTargetCS()**, and **OGRCoordinateTransformation::Transform()**.

**12.70.2.24** `int OGRPoint::WkbSize( ) const [virtual]`

Returns size of related binary representation.

This method returns the exact number of bytes required to hold the well known binary representation of this geometry object. Its computation may be slightly expensive for complex geometries.

This method relates to the **SFCOM IWks::WkbSize()** method.

This method is the same as the C function **OGR\_G\_WkbSize()** (p. ??).

#### Returns

size of binary representation in bytes.

Implements **OGRGeometry** (p. ??).

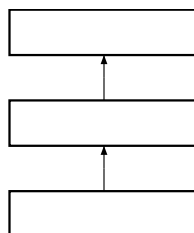
The documentation for this class was generated from the following files:

- **ogr\_geometry.h**
- **ogrpoint.cpp**

## 12.71 OGRPolygon Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRPolygon:



#### Public Member Functions

- **OGRPolygon ( )**

- Create an empty polygon.*

  - virtual const char \* **getGeometryName** () const

*Fetch WKT name for geometry type.*
- virtual **OGRWkbGeometryType** **getGeometryType** () const

*Fetch geometry type.*
- virtual **OGRGeometry** \* **clone** () const

*Make a copy of this object.*
- virtual void **empty** ()

*Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.*
- virtual OGRErr **transform** (**OGRCoordinateTransformation** \*poCT)

*Apply arbitrary coordinate transformation to geometry.*
- virtual void **flattenTo2D** ()

*Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.*
- virtual OGRBoolean **IsEmpty** () const

*Returns TRUE (non-zero) if the object has no points.*
- virtual void **segmentize** (double dfMaxLength)

*Modify the geometry such it has no segment longer then the given distance.*
- virtual double **get\_Area** () const

*Compute area of polygon.*
- virtual int **PointOnSurface** (**OGRPoint** \*poPoint) const

*This method relates to the SFCOM ISurface::get\_PointOnSurface() method.*
- virtual int **WkbSize** () const

*Returns size of related binary representation.*
- virtual OGRErr **importFromWkb** (unsigned char \*, int=-1)

*Assign geometry from well known binary data.*
- virtual OGRErr **exportToWkb** (**OGRWkbByteOrder**, unsigned char \*) const

*Convert a geometry into well known binary format.*
- virtual OGRErr **importFromWkt** (char \*\*)

*Assign geometry from well known text data.*
- virtual OGRErr **exportToWkt** (char \*\*ppsDstText) const

*Convert a geometry into well known text format.*
- virtual int **getDimension** () const

*Get the dimension of this object.*
- virtual void **getEnvelope** (**OGREnvelope** \*psEnvelope) const

*Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.*
- virtual void **getEnvelope** (**OGREnvelope3D** \*psEnvelope) const

*Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.*
- virtual OGRBoolean **Equals** (**OGRGeometry** \*) const

*Returns TRUE if two geometries are equivalent.*
- virtual void **setCoordinateDimension** (int nDimension)

*Set the coordinate dimension.*
- void **addRing** (**OGRLinearRing** \*)

*Add a ring to a polygon.*
- void **addRingDirectly** (**OGRLinearRing** \*)

*Add a ring to a polygon.*
- **OGRLinearRing** \* **getExteriorRing** ()

*Fetch reference to external polygon ring.*
- int **getNumInteriorRings** () const

*Fetch the number of internal rings.*
- **OGRLinearRing** \* **getInteriorRing** (int)

*Fetch reference to indicated internal ring.*

- virtual void **closeRings** ()

*Force rings to be closed.*

- virtual void **swapXY** ()

*Swap x and y coordinates.*

### 12.71.1 Detailed Description

Concrete class representing polygons.

Note that the OpenGIS simple features polygons consist of one outer ring, and zero or more inner rings. A polygon cannot represent disconnected regions (such as multiple islands in a political body). The **OGRMultiPolygon** (p. ??) must be used for this.

### 12.71.2 Member Function Documentation

#### 12.71.2.1 void OGRPolygon::addRing ( OGRLinearRing \* poNewRing )

Add a ring to a polygon.

If the polygon has no external ring (it is empty) this will be used as the external ring, otherwise it is used as an internal ring. The passed **OGRLinearRing** (p. ??) remains the responsibility of the caller (an internal copy is made).

This method has no SFCOM analog.

##### Parameters

<i>poNewRing</i>	ring to be added to the polygon.
------------------	----------------------------------

References OGRGeometry::getCoordinateDimension().

Referenced by clone(), OGRGeometryFactory::forceToPolygon(), OGRGeometryFactory::organizePolygons(), and OGRLayer::SetSpatialFilterRect().

#### 12.71.2.2 void OGRPolygon::addRingDirectly ( OGRLinearRing \* poNewRing )

Add a ring to a polygon.

If the polygon has no external ring (it is empty) this will be used as the external ring, otherwise it is used as an internal ring. Ownership of the passed ring is assumed by the **OGRPolygon** (p. ??), but otherwise this method operates the same as OGRPolygon::AddRing().

This method has no SFCOM analog.

##### Parameters

<i>poNewRing</i>	ring to be added to the polygon.
------------------	----------------------------------

References OGRGeometry::getCoordinateDimension().

Referenced by OGRMultiPolygon::importFromWkt(), and OGRBuildPolygonFromEdges().

#### 12.71.2.3 OGRGeometry \* OGRPolygon::clone ( ) const [virtual]

Make a copy of this object.

This method relates to the SFCOM IGeometry::clone() method.

This method is the same as the C function **OGR\_G\_Clone()** (p. ??).

**Returns**

a new object instance with the same geometry, and spatial reference system as the original.

Implements **OGRGeometry** (p. ??).

References `addRing()`, `OGRGeometry::assignSpatialReference()`, `OGRGeometry::getSpatialReference()`, and `OGRPolygon()`.

**12.71.2.4 void OGRPolygon::closeRings ( ) [virtual]**

Force rings to be closed.

If this geometry, or any contained geometries has polygon rings that are not closed, they will be closed by adding the starting point at the end.

Reimplemented from **OGRGeometry** (p. ??).

**12.71.2.5 void OGRPolygon::empty ( ) [virtual]**

Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.

This method relates to the SFCOM `IGeometry::Empty()` method.

This method is the same as the C function **OGR\_G\_Empty()** (p. ??).

Implements **OGRGeometry** (p. ??).

Referenced by `importFromWkt()`.

**12.71.2.6 OGRBoolean OGRPolygon::Equals ( OGRGeometry \* poOtherGeom ) const [virtual]**

Returns TRUE if two geometries are equivalent.

This method is the same as the C function **OGR\_G\_Equals()** (p. ??).

**Returns**

TRUE if equivalent or FALSE otherwise.

Implements **OGRGeometry** (p. ??).

References `OGRLineString::Equals()`, `getExteriorRing()`, `OGRGeometry::getGeometryType()`, `getGeometryType()`, `getInteriorRing()`, and `getNumInteriorRings()`.

**12.71.2.7 OGRErr OGRPolygon::exportToWkb ( OGRwkbByteOrder eByteOrder, unsigned char \* pabyData ) const [virtual]**

Convert a geometry into well known binary format.

This method relates to the SFCOM `IWks::ExportToWKB()` method.

This method is the same as the C function **OGR\_G\_ExportToWkb()** (p. ??).

**Parameters**

<i>eByteOrder</i>	One of <code>wkbXDR</code> or <code>wkbNDR</code> indicating MSB or LSB byte order respectively.
-------------------	--

<i>pabyData</i>	a buffer into which the binary representation is written. This buffer must be at least <b>OGRGeometry::WkbSize()</b> (p. ??) byte in size.
-----------------	--

**Returns**

Currently OGRERR\_NONE is always returned.

Implements **OGRGeometry** (p. ??).

References OGRGeometry::getCoordinateDimension(), and getGeometryType().

### 12.71.2.8 OGRErr OGRPolygon::exportToWkt ( char \*\* ppszDstText ) const [virtual]

Convert a geometry into well known text format.

This method relates to the SFCOM IWks::ExportToWKT() method.

This method is the same as the C function **OGR\_G\_ExportToWkt()** (p. ??).

**Parameters**

<i>ppszDstText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, *ppszDstText should be freed with OGRFree().
--------------------	--

**Returns**

Currently OGRERR\_NONE is always returned.

Implements **OGRGeometry** (p. ??).

References CPLCalloc(), CPLDebug(), CPLStrdup(), OGRLineString::exportToWkt(), OGRGeometry::getCoordinateDimension(), getExteriorRing(), IsEmpty(), and OGRLineString::setCoordinateDimension().

Referenced by OGRGeometryFactory::organizePolygons().

### 12.71.2.9 void OGRPolygon::flattenTo2D ( ) [virtual]

Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.

This method is the same as the C function **OGR\_G\_FlattenTo2D()** (p. ??).

Implements **OGRGeometry** (p. ??).

### 12.71.2.10 double OGRPolygon::get\_Area ( ) const [virtual]

Compute area of polygon.

The area is computed as the area of the outer ring less the area of all internal rings.

**Returns**

computed area.

Implements **OGRSurface** (p. ??).

References OGRLinearRing::get\_Area(), getExteriorRing(), getInteriorRing(), and getNumInteriorRings().

Referenced by OGRMultiPolygon::get\_Area(), and OGRGeometryFactory::organizePolygons().

### 12.71.2.11 `int OGRPolygon::getDimension ( ) const [virtual]`

Get the dimension of this object.

This method corresponds to the SFCOM IGeometry::GetDimension() method. It indicates the dimension of the object, but does not indicate the dimension of the underlying space (as indicated by **OGRGeometry::getCoordinateDimension()** (p. ??)).

This method is the same as the C function **OGR\_G\_GetDimension()** (p. ??).

#### Returns

0 for points, 1 for lines and 2 for surfaces.

Implements **OGRGeometry** (p. ??).

### 12.71.2.12 `void OGRPolygon::getEnvelope ( OGREnvelope * psEnvelope ) const [virtual]`

Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR\_G\_GetEnvelope()** (p. ??).

#### Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

Implements **OGRGeometry** (p. ??).

References OGRLineString::getEnvelope(), and IsEmpty().

### 12.71.2.13 `void OGRPolygon::getEnvelope ( OGREnvelope3D * psEnvelope ) const [virtual]`

Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.

This method is the same as the C function **OGR\_G\_GetEnvelope3D()** (p. ??).

#### Parameters

<i>psEnvelope</i>	the structure in which to place the results.
-------------------	--

#### Since

OGR 1.9.0

Implements **OGRGeometry** (p. ??).

References OGRLineString::getEnvelope(), and IsEmpty().

### 12.71.2.14 `OGRLinearRing * OGRPolygon::getExteriorRing ( )`

Fetch reference to external polygon ring.

Note that the returned ring pointer is to an internal data object of the **OGRPolygon** (p. ??). It should not be modified or deleted by the application, and the pointer is only valid till the polygon is next modified. Use the **OGRGeometry::clone()** (p. ??) method to make a separate copy within the application.

Relates to the SFCOM IPolygon::get\_ExteriorRing() method.

#### Returns

pointer to external ring. May be NULL if the **OGRPolygon** (p. ??) is empty.

Referenced by OGRGeometry::dumpReadable(), Equals(), exportToWkt(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToPolygon(), get\_Area(), and OGRGeometryFactory::organizePolygons().

12.71.2.15 `const char * OGRPolygon::getGeometryName ( ) const [virtual]`

Fetch WKT name for geometry type.

There is no SFCOM analog to this method.

This method is the same as the C function **OGR\_G\_GetGeometryName()** (p. ??).

#### Returns

name used for this geometry type in well known text format. The returned pointer is to a static internal string and should not be modified or freed.

Implements **OGRGeometry** (p. ??).

12.71.2.16 `OGRwkbGeometryType OGRPolygon::getGeometryType ( ) const [virtual]`

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the `wkbFlatten()` macro to the return result.

This method is the same as the C function **OGR\_G\_GetGeometryType()** (p. ??).

#### Returns

the geometry type code.

Implements **OGRGeometry** (p. ??).

References `OGRGeometry::getCoordinateDimension()`, `wkbPolygon`, and `wkbPolygon25D`.

Referenced by `Equals()`, and `exportToWkb()`.

12.71.2.17 `OGRLinearRing * OGRPolygon::getInteriorRing ( int iRing )`

Fetch reference to indicated internal ring.

Note that the returned ring pointer is to an internal data object of the **OGRPolygon** (p. ??). It should not be modified or deleted by the application, and the pointer is only valid till the polygon is next modified. Use the **OGRGeometry::clone()** (p. ??) method to make a separate copy within the application.

Relates to the SFCOM `IPolygon::get_InternalRing()` method.

#### Parameters

<i>iRing</i>	internal ring index from 0 to <code>getNumInternalRings()</code> - 1.
--------------	---

#### Returns

pointer to external ring. May be NULL if the **OGRPolygon** (p. ??) is empty.

Referenced by `OGRGeometry::dumpReadable()`, `Equals()`, `OGRGeometryFactory::forceToMultiLineString()`, `OGRGeometryFactory::forceToPolygon()`, and `get_Area()`.

12.71.2.18 `int OGRPolygon::getNumInteriorRings ( ) const`

Fetch the number of internal rings.

Relates to the SFCOM `IPolygon::get_NumInteriorRings()` method.

**Returns**

count of internal rings, zero or more.

Referenced by OGRGeometry::dumpReadable(), Equals(), OGRGeometryFactory::forceToMultiLineString(), OGRGeometryFactory::forceToPolygon(), and get\_Area().

#### 12.71.2.19 OGRErr OGRPolygon::importFromWkb ( unsigned char \* *pabyData*, int *nSize* = -1 ) [virtual]

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKB() method.

This method is the same as the C function **OGR\_G\_ImportFromWkb()** (p. ??).

**Parameters**

<i>pabyData</i>	the binary input data.
<i>nSize</i>	the size of pabyData in bytes, or zero if not known.

**Returns**

OGRErr\_NONE if all goes well, otherwise any of OGRErr\_NOT\_ENOUGH\_DATA, OGRErr\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRErr\_CORRUPT\_DATA may be returned.

Implements **OGRGeometry** (p. ??).

References CPLError(), VSIMalloc2(), and wkbPolygon.

#### 12.71.2.20 OGRErr OGRPolygon::importFromWkt ( char \*\* *ppszInput* ) [virtual]

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type. This method is used by the **OGRGeometryFactory** (p. ??) class, but not normally called by application code.

This method relates to the SFCOM IWks::ImportFromWKT() method.

This method is the same as the C function **OGR\_G\_ImportFromWkt()** (p. ??).

**Parameters**

<i>ppszInput</i>	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.
------------------	---

**Returns**

OGRErr\_NONE if all goes well, otherwise any of OGRErr\_NOT\_ENOUGH\_DATA, OGRErr\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRErr\_CORRUPT\_DATA may be returned.

Implements **OGRGeometry** (p. ??).

References CPLRealloc(), empty(), and OGRLineString::setPoints().

#### 12.71.2.21 OGRBoolean OGRPolygon::IsEmpty ( ) const [virtual]

Returns TRUE (non-zero) if the object has no points.

Normally this returns FALSE except between when an object is instantiated and points have been assigned.

This method relates to the SFCOM IGeometry::IsEmpty() method.



**Returns**

TRUE if object is empty, otherwise FALSE.

Implements **OGRGeometry** (p. ??).

Referenced by exportToWkt(), and getEnvelope().

**12.71.2.22** `int OGRPolygon::PointOnSurface ( OGRPoint * poPoint ) const` [virtual]

This method relates to the SFCOM ISurface::get\_PointOnSurface() method.

NOTE: Only implemented when GEOS included in build.

**Parameters**

<i>poPoint</i>	point to be set with an internal point.
----------------	---

**Returns**

OGRERR\_NONE if it succeeds or OGRERR\_FAILURE otherwise.

Implements **OGRSurface** (p. ??).

References OGRPoint::empty(), OGRPoint::getX(), OGRPoint::getY(), OGRPoint::IsEmpty(), OGR\_G\_PointOnSurface(), OGRPoint::setX(), and OGRPoint::setY().

**12.71.2.23** `void OGRPolygon::segmentize ( double dfMaxLength )` [virtual]

Modify the geometry such it has no segment longer then the given distance.

Interpolated points will have Z and M values (if needed) set to 0. Distance computation is performed in 2d only

This function is the same as the C function **OGR\_G\_Segmentize()** (p. ??)

**Parameters**

<i>dfMaxLength</i>	the maximum distance between 2 points after segmentization
--------------------	--

Reimplemented from **OGRGeometry** (p. ??).

**12.71.2.24** `void OGRPolygon::setCoordinateDimension ( int nNewDimension )` [virtual]

Set the coordinate dimension.

This method sets the explicit coordinate dimension. Setting the coordinate dimension of a geometry to 2 should zero out any existing Z values. Setting the dimension of a geometry collection will not necessarily affect the children geometries.

**Parameters**

<i>nNewDimension</i>	New coordinate dimension value, either 2 or 3.
----------------------	--

Reimplemented from **OGRGeometry** (p. ??).

References OGRGeometry::setCoordinateDimension().

**12.71.2.25** `void OGRPolygon::swapXY ( )` [virtual]

Swap x and y coordinates.

Since

OGR 1.8.0

Reimplemented from **OGRGeometry** (p. ??).

**12.71.2.26 OGRErr OGRPolygon::transform ( OGRCoordinateTransformation \* poCT ) [virtual]**

Apply arbitrary coordinate transformation to geometry.

This method will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

Note that this method does not require that the geometry already have a spatial reference system. It will be assumed that they can be treated as having the source spatial reference system of the **OGRCoordinateTransformation** (p. ??) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGRSpatialReference** (p. ??) of the **OGRCoordinateTransformation** (p. ??) will be assigned to the geometry.

This method is the same as the C function **OGR\_G\_Transform()** (p. ??).

Parameters

<i>poCT</i>	the transformation to apply.
-------------	------------------------------

Returns

OGRERR\_NONE on success or an error code.

Implements **OGRGeometry** (p. ??).

References **OGRGeometry::assignSpatialReference()**, **CPLDebug()**, **OGRCoordinateTransformation::GetTargetCS()**, and **OGRLineString::transform()**.

**12.71.2.27 int OGRPolygon::WkbSize ( ) const [virtual]**

Returns size of related binary representation.

This method returns the exact number of bytes required to hold the well known binary representation of this geometry object. Its computation may be slightly expensive for complex geometries.

This method relates to the **SFCOM IWks::WkbSize()** method.

This method is the same as the C function **OGR\_G\_WkbSize()** (p. ??).

Returns

size of binary representation in bytes.

Implements **OGRGeometry** (p. ??).

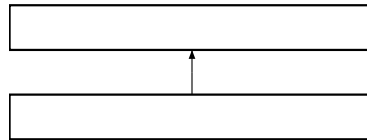
References **OGRGeometry::getCoordinateDimension()**.

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

- **ogr\_geometry.h**
- **ogrpolygon.cpp**

## 12.72 OGRProj4CT Class Reference

Inheritance diagram for OGRProj4CT:



## Public Member Functions

- virtual **OGRSpatialReference** \* **GetSourceCS** ()
- virtual **OGRSpatialReference** \* **GetTargetCS** ()
- virtual int **Transform** (int nCount, double \*x, double \*y, double \*z=NULL)
- virtual int **TransformEx** (int nCount, double \*x, double \*y, double \*z=NULL, int \*panSuccess=NULL)

## Additional Inherited Members

### 12.72.1 Member Function Documentation

#### 12.72.1.1 **OGRSpatialReference** \* **OGRProj4CT::GetSourceCS** ( ) [virtual]

Fetch internal source coordinate system.

Implements **OGRCoordinateTransformation** (p. ??).

#### 12.72.1.2 **OGRSpatialReference** \* **OGRProj4CT::GetTargetCS** ( ) [virtual]

Fetch internal target coordinate system.

Implements **OGRCoordinateTransformation** (p. ??).

#### 12.72.1.3 int **OGRProj4CT::Transform** ( int *nCount*, double \* *x*, double \* *y*, double \* *z* = NULL ) [virtual]

Transform points from source to destination space.

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

The method **TransformEx**() (p. ??) allows extended success information to be captured indicating which points failed to transform.

#### Parameters

<i>nCount</i>	number of points to transform.
<i>x</i>	array of nCount X vertices, modified in place.
<i>y</i>	array of nCount Y vertices, modified in place.
<i>z</i>	array of nCount Z vertices, modified in place.

#### Returns

TRUE on success, or FALSE if some or all points fail to transform.

Implements **OGRCoordinateTransformation** (p. ??).

References CPLMalloc(), and TransformEx().

#### 12.72.1.4 int **OGRProj4CT::TransformEx** ( int *nCount*, double \* *x*, double \* *y*, double \* *z* = NULL, int \* *panSuccess* = NULL ) [virtual]

Transform points from source to destination space.

This method is the same as the C function `OCTTransformEx()`.

## Parameters

<i>nCount</i>	number of points to transform.
<i>x</i>	array of nCount X vertices, modified in place.
<i>y</i>	array of nCount Y vertices, modified in place.
<i>z</i>	array of nCount Z vertices, modified in place.
<i>pabSuccess</i>	array of per-point flags set to TRUE if that point transforms, or FALSE if it does not.

## Returns

TRUE if some or all points transform successfully, or FALSE if if none transform.

Implements **OGRCoordinateTransformation** (p. ??).

References CPLError(), and CPLRealloc().

Referenced by Transform().

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

- ogrct.cpp

## 12.73 OGRProj4Datum Struct Reference

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

- ogr\_srs\_proj4.cpp

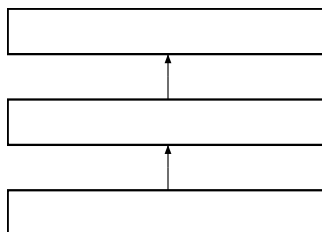
## 12.74 OGRProj4PM Struct Reference

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

- ogr\_srs\_proj4.cpp

## 12.75 OGRProxiedLayer Class Reference

Inheritance diagram for OGRProxiedLayer:



## 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 OGRErr **SetAttributeFilter** (const char \*)  
*Set a new attribute query.*
- virtual void **ResetReading** ()  
*Reset feature reading to start on the first feature.*
- virtual **OGRFeature \*** **GetNextFeature** ()  
*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** ()  
*Fetch the schema information for this 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 int **TestCapability** (const char \*)  
*Test if this layer supported the named capability.*
- 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 **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 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.*

## 12.75.1 Member Function Documentation

**12.75.1.1 OGRErr OGRProxiedLayer::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** (p. ??) for the layer will be updated to reflect the altered field. Applications should never modify the **OGRFeatureDefn** (p. ??) 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 **OLCAAlterFieldDefn** 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()** (p. ??).

### Parameters

<i>iField</i>	index of the field whose definition must be altered.
<i>poNewFieldDefn</i>	new field definition
<i>nFlags</i>	combination of ALTER_NAME_FLAG, ALTER_TYPE_FLAG and ALTER_WIDTH_PRECISION_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

Reimplemented from **OGRLayer** (p. ??).

References **OGRLayer::AlterFieldDefn()**.

**12.75.1.2 OGRErr OGRProxiedLayer::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()** (p. ??).

### Parameters

<i>poFeature</i>	the feature to write to disk.
------------------	-------------------------------

### Returns

OGRERR\_NONE on success.

Reimplemented from **OGRLayer** (p. ??).

References **OGRLayer::CreateFeature()**.

### 12.75.1.3 OGRErr OGRProxiedLayer::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** (p. ??) for the layer will be updated to reflect the new field. Applications should never modify the **OGRFeatureDefn** (p. ??) 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 existings features of the backing file/database should be updated accordingly.

This function is the same as the C function **OGR\_L\_CreateField()** (p. ??).

#### Parameters

<i>poField</i>	field definition to write to disk.
<i>bApproxOK</i>	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 from **OGRLayer** (p. ??).

References OGRLayer::CreateField().

### 12.75.1.4 OGRErr OGRProxiedLayer::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()** (p. ??) 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()** (p. ??).

#### Parameters

<i>nFID</i>	the feature id to be deleted from the layer
-------------	---

#### Returns

OGRERR\_NONE on success.

Reimplemented from **OGRLayer** (p. ??).

References OGRLayer::DeleteFeature().

### 12.75.1.5 OGRErr OGRProxiedLayer::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** (p. ??) for the layer will be updated to reflect the deleted field. Applications should never modify the **OGRFeatureDefn** (p. ??) 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 existings features of the backing file/database should be updated accordingly.

This function is the same as the C function `OGR_L_DeleteField()` (p. ??).

#### Parameters

<i>iField</i>	index of the field to delete.
---------------	-------------------------------

#### Returns

`OGRERR_NONE` on success.

#### Since

OGR 1.9.0

Reimplemented from `OGRLayer` (p. ??).

References `OGRLayer::DeleteField()`.

#### 12.75.1.6 `OGRERR OGRProxiedLayer::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()` (p. ??) 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()` (p. ??).

#### Parameters

<i>psExtent</i>	the structure in which the extent value will be returned.
<i>bForce</i>	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 from `OGRLayer` (p. ??).

References `OGRLayer::GetExtent()`.

#### 12.75.1.7 `OGRFeature * OGRProxiedLayer::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.

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

Use `OGRLayer::TestCapability(OLCRandomRead)` to establish if this layer supports efficient random access reading via **GetFeature()** (p. ??); 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 are generally considered interrupted by a **GetFeature()** (p. ??) call.

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

This method is the same as the C function **OGR\_L\_GetFeature()** (p. ??).

#### Parameters

<i>nFID</i>	the feature id of the feature to read.
-------------	--

#### Returns

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

Reimplemented from **OGRLayer** (p. ??).

References `OGRLayer::GetFeature()`.

#### 12.75.1.8 `int OGRProxiedLayer::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()** (p. ??).

#### Parameters

<i>bForce</i>	Flag indicating whether the count should be computed even if it is expensive.
---------------	---

#### Returns

feature count, -1 if count not known.

Reimplemented from **OGRLayer** (p. ??).

References `OGRLayer::GetFeatureCount()`.

#### 12.75.1.9 `const char * OGRProxiedLayer::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()** (p. ??).

#### Returns

fid column name.

Reimplemented from **OGRLayer** (p. ??).

References `OGRLayer::GetFIDColumn()`.

**12.75.1.10** `const char * OGRProxiedLayer::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()** (p. ??).

**Returns**

geometry column name.

Reimplemented from **OGRLayer** (p. ??).

References **OGRLayer::GetGeometryColumn()**.

**12.75.1.11** `OGRwkbGeometryType OGRProxiedLayer::GetGeomType ( ) [virtual]`

Return the layer geometry type.

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

This method is the same as the C function **OGR\_L\_GetGeomType()** (p. ??).

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

**Returns**

the geometry type

**Since**

OGR 1.8.0

Reimplemented from **OGRLayer** (p. ??).

References **OGRLayer::GetGeomType()**, and **wkbUnknown**.

**12.75.1.12** `OGRFeatureDefn * OGRProxiedLayer::GetLayerDefn ( ) [virtual]`

Fetch the schema information for this layer.

The returned **OGRFeatureDefn** (p. ??) is owned by the **OGRLayer** (p. ??), 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()** (p. ??).

**Returns**

feature definition.

Implements **OGRLayer** (p. ??).

References **OGRLayer::GetLayerDefn()**, and **OGRFeatureDefn::Reference()**.

**12.75.1.13** `const char * OGRProxiedLayer::GetName ( ) [virtual]`

Return the layer name.

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

This method is the same as the C function **OGR\_L\_GetName()** (p. ??).

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

#### Returns

the layer name (must not be freed)

#### Since

OGR 1.8.0

Reimplemented from **OGRLayer** (p. ??).

References **OGRLayer::GetName()**.

#### 12.75.1.14 **OGRFeature \* OGRProxiedLayer::GetNextFeature ( )** [virtual]

Fetch the next available feature from this layer.

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

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

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

This method is the same as the C function **OGR\_L\_GetNextFeature()** (p. ??).

#### Returns

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

Implements **OGRLayer** (p. ??).

References **OGRLayer::GetNextFeature()**.

#### 12.75.1.15 **OGRGeometry \* OGRProxiedLayer::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()** (p. ??).

#### Returns

spatial filter geometry.

Reimplemented from **OGRLayer** (p. ??).

References **OGRLayer::GetSpatialFilter()**.

#### 12.75.1.16 **OGRSpatialReference \* OGRProxiedLayer::GetSpatialRef ( )** [virtual]

Fetch the spatial reference system for this layer.

The returned object is owned by the **OGRLayer** (p. ??) and should not be modified or freed by the application.

This method is the same as the C function **OGR\_L\_GetSpatialRef()** (p. ??).

**Returns**

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

Reimplemented from **OGRLayer** (p. ??).

References OGRLayer::GetSpatialRef(), and OGRSpatialReference::Reference().

**12.75.1.17 OGRStyleTable \* OGRProxiedLayer::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 from **OGRLayer** (p. ??).

References OGRLayer::GetStyleTable().

**12.75.1.18 OGRErr OGRProxiedLayer::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** (p. ??) for the layer will be updated to reflect the reordering of the fields. Applications should never modify the **OGRFeatureDefn** (p. ??) 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 existings features of the backing file/database should be updated accordingly.

This function is the same as the C function **OGR\_L\_ReorderFields()** (p. ??).

**Parameters**

<i>panMap</i>	an array of <b>GetLayerDefn()</b> (p. ??)->GetFieldCount() elements which is a permutation of [0, <b>GetLayerDefn()</b> (p. ??)->GetFieldCount()-1].
---------------	--

**Returns**

OGRERR\_NONE on success.

**Since**

OGR 1.9.0

Reimplemented from **OGRLayer** (p. ??).

References OGRLayer::ReorderFields().

12.75.1.19 void OGRProxiedLayer::ResetReading ( ) [virtual]

Reset feature reading to start on the first feature.

This affects **GetNextFeature()** (p. ??).

This method is the same as the C function **OGR\_L\_ResetReading()** (p. ??).

Implements **OGRLayer** (p. ??).

References OGRLayer::ResetReading().

12.75.1.20 OGRErr OGRProxiedLayer::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()** (p. ??) 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()** (p. ??)).

This method is the same as the C function **OGR\_L\_SetAttributeFilter()** (p. ??).

#### Parameters

<i>pszQuery</i>	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 from **OGRLayer** (p. ??).

References OGRLayer::SetAttributeFilter().

12.75.1.21 OGRErr OGRProxiedLayer::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** (p. ??).

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

This method is the same as the C function **OGR\_L\_SetFeature()** (p. ??).

#### Parameters

<i>poFeature</i>	the feature to write.
------------------	-----------------------

#### Returns

OGRERR\_NONE if the operation works, otherwise an appropriate error code.

Reimplemented from **OGRLayer** (p. ??).

References OGRLayer::SetFeature().

### 12.75.1.22 OGRErr OGRProxiedLayer::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()** (p. ??) / **GetNextFeature()** (p. ??) 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()** (p. ??)

#### Parameters

<i>papszFields</i>	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 from **OGRLayer** (p. ??).

References OGRLayer::SetIgnoredFields().

### 12.75.1.23 OGRErr OGRProxiedLayer::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()** (p. ??) 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()** (p. ??) 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()** (p. ??) efficiently implemented. In all other cases the default implementation which calls **ResetReading()** (p. ??) and then calls **GetNextFeature()** (p. ??) nIndex times is used. To determine if fast seeking is available on the current layer use the **TestCapability()** (p. ??) method with a value of OLCFastSetNextByIndex.

This method is the same as the C function **OGR\_L\_SetNextByIndex()** (p. ??).

#### Parameters

<i>nIndex</i>	the index indicating how many steps into the result set to seek.
---------------	--

#### Returns

OGRErr\_NONE on success or an error code.

Reimplemented from **OGRLayer** (p. ??).

References OGRLayer::SetNextByIndex().

### 12.75.1.24 void OGRProxiedLayer::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()** (p. ??) 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()** (p. ??)) 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()** (p. ??)). In the future this may be generalized.

This method is the same as the C function **OGR\_L\_SetSpatialFilter()** (p. ??).

#### Parameters

<i>poFilter</i>	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 from **OGRLayer** (p. ??).

References **OGRLayer::SetSpatialFilter()**.

#### 12.75.1.25 void OGRProxiedLayer::SetStyleTable ( OGRStyleTable \* *poStyleTable* ) [virtual]

Set layer style table.

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

This method is the same as the C function **OGR\_L\_SetStyleTable()**.

#### Parameters

<i>poStyleTable</i>	pointer to style table to set
---------------------	-------------------------------

Reimplemented from **OGRLayer** (p. ??).

References **OGRLayer::SetStyleTable()**.

#### 12.75.1.26 void OGRProxiedLayer::SetStyleTableDirectly ( OGRStyleTable \* *poStyleTable* ) [virtual]

Set layer style table.

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

This method is the same as the C function **OGR\_L\_SetStyleTableDirectly()**.

#### Parameters

<i>poStyleTable</i>	pointer to style table to set
---------------------	-------------------------------

Reimplemented from **OGRLayer** (p. ??).

References **OGRLayer::SetStyleTableDirectly()**.

#### 12.75.1.27 OGRErr OGRProxiedLayer::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()** (p. ??)



that will ensure all data is correctly flushed.

This method is the same as the C function **OGR\_L\_SyncToDisk()** (p. ??).

#### Returns

OGRERR\_NONE if no error occurs (even if nothing is done) or an error code.

Reimplemented from **OGRLayer** (p. ??).

References **OGRLayer::SyncToDisk()**.

#### 12.75.1.28 int OGRProxiedLayer::TestCapability ( const char \* pszCap ) [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()** (p. ??) method is implemented in an optimized way for this layer, as opposed to the default implementation using **ResetReading()** (p. ??) and **GetNextFeature()** (p. ??) to find the requested feature id.
- **OLCSequentialWrite** / "SequentialWrite": TRUE if the **CreateFeature()** (p. ??) method works for this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. ??) class may returned FALSE for other layer instances that are effectively read-only.
- **OLCRandomWrite** / "RandomWrite": TRUE if the **SetFeature()** (p. ??) method is operational on this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. ??) 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** (p. ??) 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()** (p. ??)) 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()** (p. ??)) 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()** (p. ??) call efficiently, otherwise FALSE.
- **OLCCreateField** / "CreateField": TRUE if this layer can create new fields on the current layer using **CreateField()** (p. ??), otherwise FALSE.
- **OLCDeleteField** / "DeleteField": TRUE if this layer can delete existing fields on the current layer using **DeleteField()** (p. ??), otherwise FALSE.
- **OLCReorderFields** / "ReorderFields": TRUE if this layer can reorder existing fields on the current layer using **ReorderField()** (p. ??) or **ReorderFields()** (p. ??), otherwise FALSE.
- **OLCAAlterFieldDefn** / "AlterFieldDefn": TRUE if this layer can alter the definition of an existing field on the current layer using **AlterFieldDefn()** (p. ??), otherwise FALSE.
- **OLCDeleteFeature** / "DeleteFeature": TRUE if the **DeleteFeature()** (p. ??) 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()** (p. ??) method.

This method is the same as the C function **OGR\_L\_TestCapability()** (p. ??).

#### Parameters

<i>pszCap</i>	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.

Implements **OGRLayer** (p. ??).

References OGRLayer::TestCapability().

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

- ogrlayerpool.h
- ogrlayerpool.cpp

## 12.76 OGRRawPoint Class Reference

```
#include <ogr_geometry.h>
```

### 12.76.1 Detailed Description

Simple container for a position.

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

- **ogr\_geometry.h**

## 12.77 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.*

### 12.77.1 Detailed Description

Represents an operational format driver.

One **OGRSFDriver** (p. ??) 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 **OGRSFDriver-Registrar** (p. ??).

### 12.77.2 Member Function Documentation

**12.77.2.1 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()** (p. ??) 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()** (p. ??).

#### Parameters

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

#### Returns

NULL is returned on failure, or a new **OGRDataSource** (p. ??) handle on success.

References **OGRDataSource::CopyLayer()**, **CPLError()**, **CreateDataSource()**, **OGRDataSource::GetDriver()**, **OGRDataSource::GetLayer()**, **OGRDataSource::GetLayerCount()**, **OGRLayer::GetLayerDefn()**, **OGRFeatureDefn::GetName()**, **GetName()**, **OGRDataSource::SetDriver()**, and **TestCapability()**.

**12.77.2.2 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()** (p. ??) 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()** (p. ??).

**Note**

This method does **NOT** attach driver instance to the returned data source, so caller should expect that **OGRDataSource::GetDriver()** (p. ??) 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

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

## Returns

NULL is returned on failure, or a new **OGRDataSource** (p. ??) on success.

References CPLError().

Referenced by CopyDataSource(), and OGR\_Dr\_CreateDataSource().

### 12.77.2.3 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()** (p. ??) on ODrCDeleteDataSource.

This method is the same as the C function **OGR\_Dr\_DeleteDataSource()** (p. ??).

## Parameters

<i>pszDataSource</i>	the name of the datasource to delete.
----------------------	---------------------------------------

## Returns

OGRERR\_NONE on success, and OGRERR\_UNSUPPORTED\_OPERATION if this is not supported by this driver.

References CPLError().

### 12.77.2.4 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()** (p. ??).

## Returns

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

Referenced by CopyDataSource(), OGRSFDriverRegistrar::Open(), and OGRSFDriverRegistrar::RegisterDriver().

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

Attempt to open file with this driver.

This method is what **OGRSFDriverRegistrar** (p. ??) uses to implement its **Open()** (p. ??) 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** (p. ??)) 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()** (p. ??)), in a .tar/.tar.gz/.tgz archive (see **VSIInstallTarFileHandler()** (p. ??)) or on a HTTP / FTP server (see **VSIInstallCurlFileHandler()** (p. ??))

This method is the same as the C function **OGR\_Dr\_Open()** (p. ??).

#### Parameters

<i>pszName</i>	the name of the file, or data source to try and open.
<i>bUpdate</i>	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** (p. ??). This **OGRDataSource** (p. ??) should be closed by deleting the object when it is no longer needed.

Referenced by OGRSFDriverRegistrar::Open().

**12.77.2.6** `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 misspelling.

This method is the same as the C function **OGR\_Dr\_TestCapability()** (p. ??).

#### Parameters

<i>pszCapability</i>	the capability to test.
----------------------	-------------------------

#### Returns

TRUE if capability available otherwise FALSE.

Referenced by CopyDataSource().

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

- **ogrsf\_frmts.h**
- **ogrsf\_frmts.dox**
- **ogrsfdriver.cpp**

## 12.78 OGRSFDriverRegistrar Class Reference

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

## Public Member Functions

- 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** ()  
*Auto-load GDAL drivers from shared libraries.*

## 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.*

### 12.78.1 Detailed Description

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

### 12.78.2 Member Function Documentation

#### 12.78.2.1 void OGRSFDriverRegistrar::AutoLoadDrivers ( )

Auto-load GDAL drivers from shared libraries.

This function will automatically load drivers from shared libraries. It searches the "driver path" for .so (or .dll) files that start with the prefix "ogr\_X.so". It then tries to load them and then tries to call a function within them called RegisterOGRX() where the 'X' is the same as the remainder of the shared library basename, or failing that to call GDALRegisterMe().

There are a few rules for the driver path. If the GDAL\_DRIVER\_PATH environment variable is set, it is taken to be a list of directories to search separated by colons on unix, or semi-colons on Windows.

If that is not set the following defaults are used:

- Linux/Unix: <prefix>/lib/gdalplugins is searched or /usr/local/lib/gdalplugins if the install prefix is not known.
- MacOSX: <prefix>/PlugIns is searched, or /usr/local/lib/gdalplugins if the install prefix is not known. Also, the framework directory /Library/Application Support/GDAL/PlugIns is searched.

- Win32: <prefix>/lib/gdalplugins if the prefix is known (normally it is not), otherwise the gdalplugins subdirectory of the directory containing the currently running executable is used.

References CPLCalloc(), CPLDebug(), CPLFormFilename(), CPLGetBasename(), CPLGetConfigOption(), CPLGetDirname(), CPLGetExecPath(), CPLGetExtension(), CPLGetSymbol(), CSLCount(), CSLDestroy(), and VSIStatL().

Referenced by OGRRegisterAll().

#### 12.78.2.2 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()** (p. ??).

##### Parameters

<i>poDriver</i>	the driver to deregister.
-----------------	---------------------------

##### Since

GDAL 1.8.0

Referenced by OGRDeregisterDriver().

#### 12.78.2.3 OGRSFDriver \* OGRSFDriverRegistrar::GetDriver ( int *iDriver* )

Fetch the indicated driver.

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

##### Parameters

<i>iDriver</i>	the driver index, from 0 to <b>GetDriverCount()</b> (p. ??)-1.
----------------	--

##### Returns

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

Referenced by OGRGetDriver().

#### 12.78.2.4 OGRSFDriver \* OGRSFDriverRegistrar::GetDriverByName ( const char \* *pszName* )

Fetch the indicated driver.

This method is the same as the C function **OGRGetDriverByName**

##### Parameters

<i>pszName</i>	the driver name
----------------	-----------------

##### Returns

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

Referenced by OGRGetDriverByName().



**12.78.2.5** `int OGRSFDriverRegistrar::GetDriverCount ( void )`

Fetch the number of registered drivers.

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

**Returns**

the drivers count.

Referenced by OGRGetDriverCount().

**12.78.2.6** `OGRDataSource * OGRSFDriverRegistrar::GetOpenDS ( int iDS )`

Return the iDS th datasource opened.

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

**Parameters**

<i>iDS</i>	the index of the dataset to return (between 0 and <b>GetOpenDSCount()</b> (p. ??) - 1)
------------	--

Referenced by OGRGetOpenDS().

**12.78.2.7** `int OGRSFDriverRegistrar::GetOpenDSCount ( ) [inline]`

Return the number of opened datasources.

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

**Returns**

the number of opened datasources.

Referenced by OGRGetOpenDSCount().

**12.78.2.8** `OGRSFDriverRegistrar * OGRSFDriverRegistrar::GetRegistrar ( ) [static]`

Return the driver manager, creating one if none exist.

Fetch registrar.

**Returns**

the driver manager.

This static method should be used to fetch the singleton registrar. It will create a registrar if there is not already one in existence.

**Returns**

the current driver registrar.

Referenced by OGRDeregisterDriver(), OGRGetDriverByName(), OGRGetOpenDS(), OGRGetOpenDSCount(), OGRRegisterAll(), OGRRegisterDriver(), OGRReleaseDataSource(), Open(), and OGRDataSource::Release().

**12.78.2.9** `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** (p. ??) methods directly, nor do they normally need to have a pointer to an **OGRSFDriverRegistrar** (p. ??) instance.

If this method fails, **CPLGetLastErrorMsg()** (p. ??) 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 **VSIInstallZipFileHandler()** (p. ??)), in a .tar/.tar.gz/.tgz archive (see **VSIInstallTarFileHandler()** (p. ??)) or on a HTTP / FTP server (see **VSIInstallCurlFileHandler()** (p. ??))

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

#### Parameters

<i>pszName</i>	the name of the file, or data source to open. UTF-8 encoded.
<i>bUpdate</i>	FALSE for read-only access (the default) or TRUE for read-write access.
<i>ppoDriver</i>	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** (p. ??). This **OGRDataSource** (p. ??) should be closed by deleting the object when it is no longer needed.

#### Example:

```
OGRDataSource (p.??) *poDS;

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

... use the data source ...

OGRDataSource::DestroyDataSource(poDS);
```

References CPLDebug(), CPLErrorReset(), CPLGetLastErrorType(), OGRDataSource::GetDriver(), OGRSFDriver::GetName(), GetRegistrar(), OGRSFDriver::Open(), and OGRDataSource::Reference().

Referenced by OGROpen().

#### 12.78.2.10 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()** (p. ??).

#### Parameters

<i>poDriver</i>	the driver to add.
-----------------	--------------------

References CPLGetConfigOption(), CPLRealloc(), CSLDestroy(), and OGRSFDriver::GetName().

Referenced by OGRRegisterDriver().

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

- ogrsf\_frmts.h

- ogrsf\_frmts.dox
- ogrsfdriverregistrar.cpp

## 12.79 OGRSpatialReference Class Reference

```
#include <ogr_spatialref.h>
```

### Public Member Functions

- **OGRSpatialReference** (const char \*=NULL)  
*Constructor.*
- virtual ~**OGRSpatialReference** ()  
***OGRSpatialReference** (p. ??) destructor.*
- int **Reference** ()  
*Increments the reference count by one.*
- int **Dereference** ()  
*Decrements the reference count by one.*
- int **GetReferenceCount** () const  
*Fetch current reference count.*
- void **Release** ()  
*Decrements the reference count by one, and destroy if zero.*
- **OGRSpatialReference** \* **Clone** () const  
*Make a duplicate of this **OGRSpatialReference** (p. ??).*
- **OGRSpatialReference** \* **CloneGeogCS** () const  
*Make a duplicate of the GEOGCS node of this **OGRSpatialReference** (p. ??) object.*
- OGRErr **exportToWkt** (char \*\*) const  
*Convert this SRS into WKT format.*
- OGRErr **exportToPrettyWkt** (char \*\*, int=FALSE) const
- OGRErr **exportToProj4** (char \*\*) const  
*Export coordinate system in PROJ.4 format.*
- OGRErr **exportToPCI** (char \*\*, char \*\*, double \*\*) const  
*Export coordinate system in PCI projection definition.*
- OGRErr **exportToUSGS** (long \*, long \*, double \*\*, long \*) const  
*Export coordinate system in USGS GCTP projection definition.*
- OGRErr **exportToXML** (char \*\*, const char \*=NULL) const  
*Export coordinate system in XML format.*
- OGRErr **exportToPanorama** (long \*, long \*, long \*, long \*, double \*) const
- OGRErr **exportToERM** (char \*pszProj, char \*pszDatum, char \*pszUnits)
- OGRErr **exportToMlCoordSys** (char \*\*) const  
*Export coordinate system in Mapinfo style CoordSys format.*
- OGRErr **importFromWkt** (char \*\*)   
*Import from WKT string.*
- OGRErr **importFromProj4** (const char \*)  
*Import PROJ.4 coordinate string.*
- OGRErr **importFromEPSG** (int)  
*Initialize SRS based on EPSG GCS or PCS code.*
- OGRErr **importFromEPSGA** (int)  
*Initialize SRS based on EPSG GCS or PCS code.*
- OGRErr **importFromESRI** (char \*\*)   
*Import coordinate system from ESRI .prj format(s).*

- OGRErr **importFromPCI** (const char \*, const char \*=NULL, double \*=NULL)  
*Import coordinate system from PCI projection definition.*
- OGRErr **importFromUSGS** (long iProjSys, long iZone, double \*padfPrjParams, long iDatum, int nUSGS-AngleFormat=TRUE)  
*Import coordinate system from USGS projection definition.*
- OGRErr **importFromPanorama** (long, long, long, double \*)
- OGRErr **importFromOzi** (const char \*, const char \*, const char \*)
- OGRErr **importFromOzi** (const char \*const \*papszLines)
- OGRErr **importFromWMSAUTO** (const char \*pszAutoDef)  
*Initialize from WMSAUTO string.*
- OGRErr **importFromXML** (const char \*)  
*Import coordinate system from XML format (GML only currently).*
- OGRErr **importFromDict** (const char \*pszDict, const char \*pszCode)
- OGRErr **importFromURN** (const char \*)  
*Initialize from OGC URN.*
- OGRErr **importFromCRSURL** (const char \*)  
*Initialize from OGC URL.*
- OGRErr **importFromERM** (const char \*pszProj, const char \*pszDatum, const char \*pszUnits)
- OGRErr **importFromUrl** (const char \*)  
*Set spatial reference from a URL.*
- OGRErr **importFromMlCoordSys** (const char \*)  
*Import Mapinfo style CoordSys definition.*
- OGRErr **morphToESRI** ()  
*Convert in place to ESRI WKT format.*
- OGRErr **morphFromESRI** ()  
*Convert in place from ESRI WKT format.*
- OGRErr **Validate** ()  
*Validate SRS tokens.*
- OGRErr **StripCTParms** (OGR\_SRSNode \*=NULL)  
*Strip OGC CT Parameters.*
- OGRErr **StripVertical** ()  
*Convert a compound cs into a horizontal CS.*
- OGRErr **FixupOrdering** ()  
*Correct parameter ordering to match CT Specification.*
- OGRErr **Fixup** ()  
*Fixup as needed.*
- int **EPSGTreatsAsLatLong** ()  
*This method returns TRUE if EPSG feels this geographic coordinate system should be treated as having lat/long coordinate ordering.*
- int **EPSGTreatsAsNorthingEasting** ()  
*This method returns TRUE if EPSG feels this projected coordinate system should be treated as having northing/easting coordinate ordering.*
- const char \* **GetAxis** (const char \*pszTargetKey, int iAxis, OGRAxisOrientation \*peOrientation) const  
*Fetch the orientation of one axis.*
- OGRErr **SetAxes** (const char \*pszTargetKey, const char \*pszXAxisName, OGRAxisOrientation eXAxisOrientation, const char \*pszYAxisName, OGRAxisOrientation eYAxisOrientation)  
*Set the axes for a coordinate system.*
- void **SetRoot** (OGR\_SRSNode \*)  
*Set the root SRS node.*
- OGR\_SRSNode \* **GetAttrNode** (const char \*)  
*Find named node in tree.*
- const char \* **GetAttrValue** (const char \*, int=0) const

- Fetch indicated attribute of named node.*

  - OGRErr **SetNode** (const char \*, const char \*)

*Set attribute value in spatial reference.*
- OGRErr **SetLinearUnitsAndUpdateParameters** (const char \*pszName, double dfInMeters)

*Set the linear units for the projection.*
- OGRErr **SetLinearUnits** (const char \*pszName, double dfInMeters)

*Set the linear units for the projection.*
- OGRErr **SetTargetLinearUnits** (const char \*pszTargetKey, const char \*pszName, double dfInMeters)

*Set the linear units for the projection.*
- double **GetLinearUnits** (char \*\*=NULL) const

*Fetch linear projection units.*
- double **GetTargetLinearUnits** (const char \*pszTargetKey, char \*\*ppszRetName=NULL) const

*Fetch linear units for target.*
- OGRErr **SetAngularUnits** (const char \*pszName, double dfInRadians)

*Set the angular units for the geographic coordinate system.*
- double **GetAngularUnits** (char \*\*=NULL) const

*Fetch angular geographic coordinate system units.*
- double **GetPrimeMeridian** (char \*\*=NULL) const

*Fetch prime meridian info.*
- int **IsGeographic** () const

*Check if geographic coordinate system.*
- int **IsProjected** () const

*Check if projected coordinate system.*
- int **IsGeocentric** () const

*Check if geocentric coordinate system.*
- int **IsLocal** () const

*Check if local coordinate system.*
- int **IsVertical** () const

*Check if vertical coordinate system.*
- int **IsCompound** () const

*Check if coordinate system is compound.*
- int **IsSameGeogCS** (const OGRSpatialReference \*) const

*Do the GeogCS'es match?*
- int **IsSameVertCS** (const OGRSpatialReference \*) const

*Do the VertCS'es match?*
- int **IsSame** (const OGRSpatialReference \*) const

*Do these two spatial references describe the same system ?*
- void **Clear** ()

*Wipe current definition.*
- OGRErr **SetLocalCS** (const char \*)

*Set the user visible LOCAL\_CS name.*
- OGRErr **SetProjCS** (const char \*)

*Set the user visible PROJCS name.*
- OGRErr **SetProjection** (const char \*)

*Set a projection name.*
- OGRErr **SetGeocCS** (const char \*pszGeocName)

*Set the user visible GEOCCS name.*
- OGRErr **SetGeogCS** (const char \*pszGeogName, const char \*pszDatumName, const char \*pszEllipsoidName, double dfSemiMajor, double dfInvFlattening, const char \*pszPMName=NULL, double dfPMOffset=0.-0, const char \*pszUnits=NULL, double dfConvertToRadians=0.0)

*Set geographic coordinate system.*

- OGRErr **SetWellKnownGeogCS** (const char \*)  
*Set a GeogCS based on well known name.*
- OGRErr **CopyGeogCSFrom** (const **OGRSpatialReference** \*poSrcSRS)  
*Copy GEOGCS from another **OGRSpatialReference** (p. ??).*
- OGRErr **SetVertCS** (const char \*pszVertCSName, const char \*pszVertDatumName, int nVertDatumClass=2005)  
*Set the user visible VERT\_CS name.*
- OGRErr **SetCompoundCS** (const char \*pszName, const **OGRSpatialReference** \*poHorizSRS, const **OGRSpatialReference** \*poVertSRS)  
*Setup a compound coordinate system.*
- OGRErr **SetFromUserInput** (const char \*)  
*Set spatial reference from various text formats.*
- OGRErr **SetTOWGS84** (double, double, double, double=0.0, double=0.0, double=0.0, double=0.0)  
*Set the Bursa-Wolf conversion to WGS84.*
- OGRErr **GetTOWGS84** (double \*padfCoef, int nCoeff=7) const  
*Fetch TOWGS84 parameters, if available.*
- double **GetSemiMajor** (OGRErr \*p=NULL) const  
*Get spheroid semi major axis.*
- double **GetSemiMinor** (OGRErr \*p=NULL) const  
*Get spheroid semi minor axis.*
- double **GetInvFlattening** (OGRErr \*p=NULL) const  
*Get spheroid inverse flattening.*
- OGRErr **SetAuthority** (const char \*pszTargetKey, const char \*pszAuthority, int nCode)  
*Set the authority for a node.*
- OGRErr **AutoidentifyEPSG** ()  
*Set EPSG authority info if possible.*
- const char \* **GetAuthorityCode** (const char \*pszTargetKey) const  
*Get the authority code for a node.*
- const char \* **GetAuthorityName** (const char \*pszTargetKey) const  
*Get the authority name for a node.*
- const char \* **GetExtension** (const char \*pszTargetKey, const char \*pszName, const char \*pszDefault=NULL) const  
*Fetch extension value.*
- OGRErr **SetExtension** (const char \*pszTargetKey, const char \*pszName, const char \*pszValue)  
*Set extension value.*
- int **FindProjParm** (const char \*pszParameter, const **OGR\_SRSNode** \*poPROJCS=NULL) const  
*Return the child index of the named projection parameter on its parent PROJCS node.*
- OGRErr **SetProjParm** (const char \*, double)  
*Set a projection parameter value.*
- double **GetProjParm** (const char \*, double=0.0, OGRErr \*p=NULL) const  
*Fetch a projection parameter value.*
- OGRErr **SetNormProjParm** (const char \*, double)  
*Set a projection parameter with a normalized value.*
- double **GetNormProjParm** (const char \*, double=0.0, OGRErr \*p=NULL) const  
*Fetch a normalized projection parameter value.*
- OGRErr **SetACEA** (double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetAE** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetBonne** (double dfStdP1, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetCEA** (double dfStdP1, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetCS** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)

- OGRErr **SetEC** (double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetEckert** (int nVariation, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetEquirectangular** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetEquirectangular2** (double dfCenterLat, double dfCenterLong, double dfPseudoStdParallel1, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetGEOS** (double dfCentralMeridian, double dfSatelliteHeight, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetGH** (double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetIGH** ()
- OGRErr **SetGS** (double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetGaussSchreiberTMercator** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetGnomonic** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetHOM** (double dfCenterLat, double dfCenterLong, double dfAzimuth, double dfRectToSkew, double dfScale, double dfFalseEasting, double dfFalseNorthing)  
*Set a Hotine Oblique Mercator projection using azimuth angle.*
- OGRErr **SetHOM2PNO** (double dfCenterLat, double dfLat1, double dfLong1, double dfLat2, double dfLong2, double dfScale, double dfFalseEasting, double dfFalseNorthing)  
*Set a Hotine Oblique Mercator projection using two points on projection centerline.*
- OGRErr **SetHOMAC** (double dfCenterLat, double dfCenterLong, double dfAzimuth, double dfRectToSkew, double dfScale, double dfFalseEasting, double dfFalseNorthing)  
*Set an Hotine Oblique Mercator Azimuth Center projection using azimuth angle.*
- OGRErr **SetIWMPolyconic** (double dfLat1, double dfLat2, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetKrovak** (double dfCenterLat, double dfCenterLong, double dfAzimuth, double dfPseudoStdParallelLat, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetLAEA** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetLCC** (double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetLCC1SP** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetLCCB** (double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetMC** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetMercator** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetMollweide** (double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetNZMG** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetOS** (double dfOriginLat, double dfCMeridian, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetOrthographic** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetPolyconic** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetPS** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetRobinson** (double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetSinusoidal** (double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetStereographic** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)

- OGRErr **SetSOC** (double dfLatitudeOfOrigin, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetTM** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetTMVariant** (const char \*pszVariantName, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetTMG** (double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetTMSO** (double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetTPED** (double dfLat1, double dfLong1, double dfLat2, double dfLong2, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetVDG** (double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetUTM** (int nZone, int bNorth=TRUE)  
*Set UTM projection definition.*
- int **GetUTMZone** (int \*pbNorth=NULL) const  
*Get utm zone information.*
- OGRErr **SetWagner** (int nVariation, double dfCenterLat, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **SetStatePlane** (int nZone, int bNAD83=TRUE, const char \*pszOverrideUnitName=NULL, double dfOverrideUnit=0.0)  
*Set State Plane projection definition.*

## Static Public Member Functions

- static void **DestroySpatialReference** (OGRSpatialReference \*poSRS)  
*OGRSpatialReference (p. ??) destructor.*

### 12.79.1 Detailed Description

This class represents a OpenGIS Spatial Reference System, and contains methods for converting between this object organization and well known text (WKT) format. This object is reference counted as one instance of the object is normally shared between many **OGRGeometry** (p. ??) objects.

Normally application code can fetch needed parameter values for this SRS using **GetAttrValue()** (p. ??), but in special cases the underlying parse tree (or **OGR\_SRSNode** (p. ??) objects) can be accessed more directly.

See the `tutorial` for more information on how to use this class.

### 12.79.2 Constructor & Destructor Documentation

#### 12.79.2.1 OGRSpatialReference::OGRSpatialReference ( const char \* pszWKT = NULL )

Constructor.

This constructor takes an optional string argument which if passed should be a WKT representation of an SRS. Passing this is equivalent to not passing it, and then calling **importFromWkt()** (p. ??) with the WKT string.

Note that newly created objects are given a reference count of one.

The C function **OSRNewSpatialReference()** (p. ??) does the same thing as this constructor.

Parameters

<i>pszWKT</i>	well known text definition to which the object should be initialized, or NULL (the default).
---------------	--



References `importFromWkt()`.

#### 12.79.2.2 OGRSpatialReference::~~OGRSpatialReference ( ) [virtual]

**OGRSpatialReference** (p. ??) destructor.

The C function **OSRDestroySpatialReference()** (p. ??) does the same thing as this method. Preferred C++ method : **OGRSpatialReference::DestroySpatialReference()** (p. ??)

#### Deprecated

### 12.79.3 Member Function Documentation

#### 12.79.3.1 OGRErr OGRSpatialReference::AutoidentifyEPSG ( )

Set EPSG authority info if possible.

This method inspects a WKT definition, and adds EPSG authority nodes where an aspect of the coordinate system can be easily and safely corresponded with an EPSG identifier. In practice, this method will evolve over time. In theory it can add authority nodes for any object (ie. spheroid, datum, GEOGCS, units, and PROJCS) that could have an authority node. Mostly this is useful to inserting appropriate PROJCS codes for common formulations (like UTM n WGS84).

If it success the **OGRSpatialReference** (p. ??) is updated in place, and the method return `OGRErr_NONE`. If the method fails to identify the general coordinate system `OGRErr_UNSUPPORTED_SRS` is returned but no error message is posted via **CPL::Error()** (p. ??).

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

#### Returns

`OGRErr_NONE` or `OGRErr_UNSUPPORTED_SRS`.

References `GetAuthorityCode()`, `GetAuthorityName()`, `GetUTMZone()`, `IsGeographic()`, `IsProjected()`, and `SetAuthority()`.

#### 12.79.3.2 void OGRSpatialReference::Clear ( )

Wipe current definition.

Returns **OGRSpatialReference** (p. ??) to a state with no definition, as it exists when first created. It does not affect reference counts.

Referenced by `CopyGeogCSFrom()`, `importFromCRSURL()`, `importFromERM()`, `importFromOzi()`, `importFromPanorama()`, `importFromPCI()`, `importFromProj4()`, `importFromURN()`, `importFromWkt()`, `importFromWMSAUTO()`, `importFromXML()`, `SetCompoundCS()`, `SetFromUserInput()`, `SetGeogCS()`, `SetStatePlane()`, and `SetVertCS()`.

#### 12.79.3.3 OGRSpatialReference \* OGRSpatialReference::Clone ( ) const

Make a duplicate of this **OGRSpatialReference** (p. ??).

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

**Returns**

a new SRS, which becomes the responsibility of the caller.

References OGR\_SRSNode::Clone().

Referenced by exportToPrettyWkt().

### 12.79.3.4 OGRSpatialReference \* OGRSpatialReference::CloneGeogCS ( ) const

Make a duplicate of the GEOGCS node of this **OGRSpatialReference** (p. ??) object.

**Returns**

a new SRS, which becomes the responsibility of the caller.

References OGR\_SRSNode::AddChild(), OGR\_SRSNode::Clone(), CPLAtof(), GetAttrNode(), IsGeocentric(), SetAngularUnits(), and SetRoot().

Referenced by morphFromESRI().

### 12.79.3.5 OGRErr OGRSpatialReference::CopyGeogCSFrom ( const OGRSpatialReference \* poSrcSRS )

Copy GEOGCS from another **OGRSpatialReference** (p. ??).

The GEOGCS information is copied into this **OGRSpatialReference** (p. ??) from another. If this object has a PROJCS root already, the GEOGCS is installed within it, otherwise it is installed as the root.

**Parameters**

<i>poSrcSRS</i>	the spatial reference to copy the GEOGCS information from.
-----------------	--

**Returns**

OGRERR\_NONE on success or an error code.

References Clear(), OGR\_SRSNode::Clone(), OGR\_SRSNode::DestroyChild(), OGR\_SRSNode::FindChild(), GetAttrNode(), OGR\_SRSNode::InsertChild(), IsGeocentric(), and SetRoot().

Referenced by importFromERM(), importFromESRI(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), morphFromESRI(), SetGeogCS(), and SetWellKnownGeogCS().

### 12.79.3.6 int OGRSpatialReference::Dereference ( )

Decrements the reference count by one.

The method does the same thing as the C function **OSRDereference()** (p. ??).

**Returns**

the updated reference count.

References CPLDebug().

Referenced by Release().

### 12.79.3.7 void OGRSpatialReference::DestroySpatialReference ( OGRSpatialReference \* poSRS ) [static]

**OGRSpatialReference** (p. ??) destructor.

This static method will destroy a **OGRSpatialReference** (p. ??). It is equivalent to calling delete on the object, but it ensures that the deallocation is properly executed within the OGR libraries heap on platforms where this can matter (win32).

This function is the same as **OSRDestroySpatialReference()** (p. ??)

## Parameters

<i>poSRS</i>	the object to delete
--------------	----------------------

## Since

GDAL 1.7.0

## 12.79.3.8 int OGRSpatialReference::EPSGTreatsAsLatLong ( )

This method returns TRUE if EPSG feels this geographic coordinate system should be treated as having lat/long coordinate ordering.

Currently this returns TRUE for all geographic coordinate systems with an EPSG code set, and AXIS values set defining it as lat, long. Note that coordinate systems with an EPSG code and no axis settings will be assumed to not be lat/long.

FALSE will be returned for all coordinate systems that are not geographic, or that do not have an EPSG code set.

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

## Returns

TRUE or FALSE.

References `GetAttrNode()`, `GetAuthorityName()`, `OGR_SRSNode::GetChild()`, `OGR_SRSNode::GetChildCount()`, `OGR_SRSNode::GetValue()`, and `IsGeographic()`.

## 12.79.3.9 int OGRSpatialReference::EPSGTreatsAsNorthingEasting ( )

This method returns TRUE if EPSG feels this projected coordinate system should be treated as having northing/easting coordinate ordering.

Currently this returns TRUE for all projected coordinate systems with an EPSG code set, and AXIS values set defining it as northing, easting.

FALSE will be returned for all coordinate systems that are not projected, or that do not have an EPSG code set.

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

## Returns

TRUE or FALSE.

## Since

OGR 1.10.0

References `GetAttrNode()`, `GetAuthorityName()`, `OGR_SRSNode::GetChild()`, `OGR_SRSNode::GetChildCount()`, `OGR_SRSNode::GetValue()`, and `IsProjected()`.

Referenced by `importFromEPSG()`.

12.79.3.10 OGRErr OGRSpatialReference::exportToERM ( char \* *pszProj*, char \* *pszDatum*, char \* *pszUnits* )

Convert coordinate system to ERMapper format.

## Parameters

<i>pszProj</i>	32 character buffer to receive projection name.
<i>pszDatum</i>	32 character buffer to receive datum name.
<i>pszUnits</i>	32 character buffer to receive units name.

## Returns

OGRERR\_NONE on success, OGRERR\_SRS\_UNSUPPORTED if not translation is found, or OGRERR\_FAILURE on other failures.

References `GetAttrValue()`, `GetAuthorityCode()`, `GetAuthorityName()`, `GetLinearUnits()`, `GetUTMZone()`, `importFromDict()`, `IsGeographic()`, and `IsProjected()`.

### 12.79.3.11 OGRErr OGRSpatialReference::exportToMCoordSys ( char \*\* ppszResult ) const

Export coordinate system in Mapinfo style CoordSys format.

Note that the returned WKT string should be freed with `OGRFree()` or `CPLFree()` when no longer needed. It is the responsibility of the caller.

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

## Parameters

<i>ppszResult</i>	pointer to which dynamically allocated Mapinfo CoordSys definition will be assigned.
-------------------	--

## Returns

OGRERR\_NONE on success, OGRERR\_FAILURE on failure, OGRERR\_UNSUPPORTED\_OPERATION if MITAB library was not linked in.

References `CPLError()`.

### 12.79.3.12 OGRErr OGRSpatialReference::exportToPanorama ( long \* piProjSys, long \* piDatum, long \* piEllips, long \* piZone, double \* padfPrjParams ) const

Export coordinate system in "Panorama" GIS projection definition.

This method is the equivalent of the C function `OSRExportToPanorama()`.

## Parameters

<i>piProjSys</i>	Pointer to variable, where the projection system code will be returned.
<i>piDatum</i>	Pointer to variable, where the coordinate system code will be returned.
<i>piEllips</i>	Pointer to variable, where the spheroid code will be returned.
<i>piZone</i>	Pointer to variable, where the zone for UTM projection system will be returned.
<i>padfPrjParams</i>	an existing 7 double buffer into which the projection parameters will be placed. See <b>importFromPanorama()</b> (p. ??) for the list of parameters.

## Returns

OGRERR\_NONE on success or an error code on failure.

References `CPLDebug()`, `GetAttrValue()`, `GetInvFlattening()`, `GetNormProjParm()`, `GetSemiMajor()`, `GetUTMZone()`, and `IsLocal()`.

**12.79.3.13** `OGRERR OGRSpatialReference::exportToPCI ( char ** ppszProj, char ** ppszUnits, double ** ppadfPrjParams ) const`

Export coordinate system in PCI projection definition.

Converts the loaded coordinate reference system into PCI projection definition to the extent possible. The strings returned in *ppszProj*, *ppszUnits* and *ppadfPrjParams* array should be deallocated by the caller with `CPLFree()` when no longer needed.

LOCAL\_CS coordinate systems are not translatable. An empty string will be returned along with `OGRERR_NONE`.

This method is the equivalent of the C function **OSRExportToPCI()** (p. ??).

#### Parameters

<i>ppszProj</i>	pointer to which dynamically allocated PCI projection definition will be assigned.
<i>ppszUnits</i>	pointer to which dynamically allocated units definition will be assigned.
<i>ppadfPrjParams</i>	pointer to which dynamically allocated array of 17 projection parameters will be assigned. See <b>importFromPCI()</b> (p. ??) for the list of parameters.

#### Returns

`OGRERR_NONE` on success or an error code on failure.

References `CPLAtof()`, `CPLDebug()`, `CPLMalloc()`, `CPLPrintInt32()`, `CPLPrintStringFill()`, `CPLStrdup()`, `CSLCount()`, `CSLDestroy()`, `GetAttrNode()`, `GetAttrValue()`, `GetAuthorityCode()`, `GetAuthorityName()`, `OGR_SRSNode::GetChild()`, `OGR_SRSNode::GetChildCount()`, `GetInvFlattening()`, `GetLinearUnits()`, `GetNormProjParm()`, `GetSemiMajor()`, `GetTOWGS84()`, `GetUTMZone()`, `OGR_SRSNode::GetValue()`, and `IsLocal()`.

**12.79.3.14** `OGRERR OGRSpatialReference::exportToPrettyWkt ( char ** ppszResult, int bSimplify = FALSE ) const`

Convert this SRS into a nicely formatted WKT string for display to a person.

Note that the returned WKT string should be freed with `OGRFree()` or `CPLFree()` when no longer needed. It is the responsibility of the caller.

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

#### Parameters

<i>ppszResult</i>	the resulting string is returned in this pointer.
<i>bSimplify</i>	TRUE if the AXIS, AUTHORITY and EXTENSION nodes should be stripped off

#### Returns

currently `OGRERR_NONE` is always returned, but the future it is possible error conditions will develop.

References `Clone()`, `CPLStrdup()`, and `OGR_SRSNode::StripNodes()`.

**12.79.3.15** `OGRERR OGRSpatialReference::exportToProj4 ( char ** ppszProj4 ) const`

Export coordinate system in PROJ.4 format.

Converts the loaded coordinate reference system into PROJ.4 format to the extent possible. The string returned in *ppszProj4* should be deallocated by the caller with `CPLFree()` when no longer needed.

LOCAL\_CS coordinate systems are not translatable. An empty string will be returned along with `OGRERR_NONE`.

Special processing for Transverse Mercator with GDAL >= 1.10 and PROJ >= 4.8 : if the `OSR_USE_ETMERC` configuration option is set to YES, the PROJ.4 definition built from the SRS will use the 'etmerc' projection method, rather than the default 'tmerc'. This will give better accuracy (at the expense of computational speed) when reprojection occurs near the edges of the validity area for the projection.

This method is the equivalent of the C function **OSRExportToProj4()** (p. ??).

## Parameters

<i>ppszProj4</i>	pointer to which dynamically allocated PROJ.4 definition will be assigned.
------------------	--

## Returns

OGRERR\_NONE on success or an error code on failure.

References CPLAtof(), CPLError(), CPLGetConfigOption(), CPLStrdup(), CSLTestBoolean(), GetAttrNode(), GetAttrValue(), GetAuthorityCode(), GetAuthorityName(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), GetExtension(), GetInvFlattening(), GetLinearUnits(), OGR\_SRSNode::GetNode(), GetNormProjParm(), GetSemiMajor(), GetSemiMinor(), GetUTMZone(), OGR\_SRSNode::GetValue(), IsGeocentric(), and IsGeographic().

### 12.79.3.16 OGRErr OGRSpatialReference::exportToUSGS ( long \* *piProjSys*, long \* *piZone*, double \*\* *ppadfPrjParams*, long \* *piDatum* ) const

Export coordinate system in USGS GCTP projection definition.

This method is the equivalent of the C function **OSRExportToUSGS()** (p. ??).

## Parameters

<i>piProjSys</i>	Pointer to variable, where the projection system code will be returned.
<i>piZone</i>	Pointer to variable, where the zone for UTM and State Plane projection systems will be returned.
<i>ppadfPrjParams</i>	Pointer to which dynamically allocated array of 15 projection parameters will be assigned. See <b>importFromUSGS()</b> (p. ??) for the list of parameters. Caller responsible to free this array.
<i>piDatum</i>	Pointer to variable, where the datum code will be returned.

## Returns

OGRERR\_NONE on success or an error code on failure.

References CPLDebug(), CPLDecToPackedDMS(), CPLMalloc(), GetAttrValue(), GetInvFlattening(), GetNormProjParm(), GetSemiMajor(), GetUTMZone(), and IsLocal().

### 12.79.3.17 OGRErr OGRSpatialReference::exportToWkt ( char \*\* *ppszResult* ) const

Convert this SRS into WKT format.

Note that the returned WKT string should be freed with OGRFree() or CPLFree() when no longer needed. It is the responsibility of the caller.

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

## Parameters

<i>ppszResult</i>	the resulting string is returned in this pointer.
-------------------	---

## Returns

currently OGRERR\_NONE is always returned, but the future it is possible error conditions will develop.

References CPLStrdup(), and OGR\_SRSNode::exportToWkt().

Referenced by morphFromESRI().

**12.79.3.18** `OGRErr OGRSpatialReference::exportToXML ( char ** ppszRawXML, const char * pszDialect = NULL ) const`

Export coordinate system in XML format.

Converts the loaded coordinate reference system into XML format to the extent possible. The string returned in *ppszRawXML* should be deallocated by the caller with `CPLFree()` when no longer needed.

LOCAL\_CS coordinate systems are not translatable. An empty string will be returned along with `OGRErr_NONE`.

This method is the equivalent of the C function **OSRExportToXML()** (p. ??).

#### Parameters

<i>ppszRawXML</i>	pointer to which dynamically allocated XML definition will be assigned.
<i>pszDialect</i>	currently ignored. The dialect used is GML based.

#### Returns

`OGRErr_NONE` on success or an error code on failure.

References `CPLDestroyXMLNode()`, `CPLSerializeXMLTree()`, `IsGeographic()`, and `IsProjected()`.

**12.79.3.19** `int OGRSpatialReference::FindProjParm ( const char * pszParameter, const OGR_SRSNode * poPROJCS = NULL ) const`

Return the child index of the named projection parameter on its parent PROJCS node.

#### Parameters

<i>pszParameter</i>	projection parameter to look for
<i>poPROJCS</i>	projection CS node to look in. If NULL is passed, the PROJCS node of the SpatialReference object will be searched.

#### Returns

the child index of the named projection parameter. -1 on failure

References `GetAttrNode()`, `OGR_SRSNode::GetChild()`, `OGR_SRSNode::GetChildCount()`, and `OGR_SRSNode::GetValue()`.

Referenced by `GetProjParm()`, and `morphToESRI()`.

**12.79.3.20** `OGRErr OGRSpatialReference::Fixup ( )`

Fixup as needed.

Some mechanisms to create WKT using **OGRSpatialReference** (p. ??), and some imported WKT, are not valid according to the OGC CT specification. This method attempts to fill in any missing defaults that are required, and fixup ordering problems (using **OSRFixupOrdering()** (p. ??)) so that the resulting WKT is valid.

This method should be expected to evolve over time to as problems are discovered. The following are among the fixup actions this method will take:

- Fixup the ordering of nodes to match the BNF WKT ordering, using the **FixupOrdering()** (p. ??) method.
- Add missing linear or angular units nodes.

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



**Returns**

OGRERR\_NONE on success or an error code if something goes wrong.

References CPLAtof(), OGR\_SRSNode::FindChild(), FixupOrdering(), GetAttrNode(), SetAngularUnits(), and SetLinearUnits().

Referenced by morphToESRI().

**12.79.3.21 OGRErr OGRSpatialReference::FixupOrdering ( )**

Correct parameter ordering to match CT Specification.

Some mechanisms to create WKT using **OGRSpatialReference** (p. ??), and some imported WKT fail to maintain the order of parameters required according to the BNF definitions in the OpenGIS SF-SQL and CT Specifications. This method attempts to massage things back into the required order.

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

**Returns**

OGRERR\_NONE on success or an error code if something goes wrong.

References OGR\_SRSNode::FixupOrdering().

Referenced by Fixup(), importFromEPSGA(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromUSGS(), and morphFromESRI().

**12.79.3.22 double OGRSpatialReference::GetAngularUnits ( char \*\* ppszName = NULL ) const**

Fetch angular geographic coordinate system units.

If no units are available, a value of "degree" and SRS\_UA\_DEGREE\_CONV will be assumed. This method only checks directly under the GEOGCS node for units.

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

**Parameters**

<i>ppszName</i>	a pointer to be updated with the pointer to the units name. The returned value remains internal to the <b>OGRSpatialReference</b> (p. ??) and shouldn't be freed, or modified. It may be invalidated on the next <b>OGRSpatialReference</b> (p. ??) call.
-----------------	---

**Returns**

the value to multiply by angular distances to transform them to radians.

References CPLAtof(), GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), and OGR\_SRSNode::GetValue().

Referenced by morphToESRI().

**12.79.3.23 OGR\_SRSNode \* OGRSpatialReference::GetAttrNode ( const char \* pszNodePath )**

Find named node in tree.

This method does a pre-order traversal of the node tree searching for a node with this exact value (case insensitive), and returns it. Leaf nodes are not considered, under the assumption that they are just attribute value nodes.

If a node appears more than once in the tree (such as UNIT for instance), the first encountered will be returned. Use GetNode() on a subtree to be more specific.

## Parameters

<i>pszNodePath</i>	the name of the node to search for. May contain multiple components such as "GEOGCS UNIT".
--------------------	--

## Returns

a pointer to the node found, or NULL if none.

References CSLCount(), CSLDestroy(), and OGR\_SRSNode::GetNode().

Referenced by CloneGeogCS(), CopyGeogCSFrom(), EPSGTreatsAsLatLong(), EPSGTreatsAsNorthingEasting(), exportToPCI(), exportToProj4(), FindProjParm(), Fixup(), GetAngularUnits(), GetAttrValue(), GetInvFlattening(), GetPrimeMeridian(), GetProjParm(), GetSemiMajor(), GetTargetLinearUnits(), GetTOWGS84(), importFromEPSG(), importFromESRI(), importFromProj4(), IsGeographic(), IsProjected(), IsSame(), IsVertical(), morphFromESRI(), morphToESRI(), SetAngularUnits(), SetAuthority(), SetGeocCS(), SetGeogCS(), SetLinearUnitsAndUpdateParameters(), SetLocalCS(), SetProjCS(), SetProjection(), SetProjParm(), SetStatePlane(), SetTargetLinearUnits(), SetTOWGS84(), and SetVertCS().

**12.79.3.24** `const char * OGRSpatialReference::GetAttrValue ( const char * pszNodeName, int iAttr = 0 ) const`

Fetch indicated attribute of named node.

This method uses **GetAttrNode()** (p. ??) to find the named node, and then extracts the value of the indicated child. Thus a call to `GetAttrValue("UNIT",1)` would return the second child of the UNIT node, which is normally the length of the linear unit in meters.

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

## Parameters

<i>pszNodeName</i>	the tree node to look for (case insensitive).
<i>iAttr</i>	the child of the node to fetch (zero based).

## Returns

the requested value, or NULL if it fails for any reason.

References GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), and OGR\_SRSNode::GetValue().

Referenced by exportToERM(), exportToPanorama(), exportToPCI(), exportToProj4(), exportToUSGS(), GetUTMZone(), IsSame(), IsSameGeogCS(), IsSameVertCS(), morphFromESRI(), morphToESRI(), and SetUTM().

**12.79.3.25** `const char * OGRSpatialReference::GetAuthorityCode ( const char * pszTargetKey ) const`

Get the authority code for a node.

This method is used to query an AUTHORITY[] node from within the WKT tree, and fetch the code value.

While in theory values may be non-numeric, for the EPSG authority all code values should be integral.

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

## Parameters

<i>pszTargetKey</i>	the partial or complete path to the node to get an authority from. ie. "PROJCS", "GEOGCS", "GEOGCS UNIT" or NULL to search for an authority node on the root element.
---------------------	---

**Returns**

value code from authority node, or NULL on failure. The value returned is internal and should not be freed or modified.

References OGR\_SRSNode::FindChild(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), and OGR\_SRSNode::GetValue().

Referenced by AutoidentifyEPSG(), exportToERM(), exportToPCI(), exportToProj4(), and morphToESRI().

### 12.79.3.26 `const char * OGRSpatialReference::GetAuthorityName ( const char * pszTargetKey ) const`

Get the authority name for a node.

This method is used to query an AUTHORITY[] node from within the WKT tree, and fetch the authority name value.

The most common authority is "EPSG".

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

**Parameters**

<i>pszTargetKey</i>	the partial or complete path to the node to get an authority from. ie. "PROJCS", "GEOGCS", "GEOGCS UNIT" or NULL to search for an authority node on the root element.
---------------------	---

**Returns**

value code from authority node, or NULL on failure. The value returned is internal and should not be freed or modified.

References OGR\_SRSNode::FindChild(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), and OGR\_SRSNode::GetValue().

Referenced by AutoidentifyEPSG(), EPSGTreatsAsLatLong(), EPSGTreatsAsNorthingEasting(), exportToERM(), exportToPCI(), exportToProj4(), importFromEPSGA(), and morphToESRI().

### 12.79.3.27 `const char * OGRSpatialReference::GetAxis ( const char * pszTargetKey, int iAxis, OGRAxisOrientation * peOrientation ) const`

Fetch the orientation of one axis.

Fetches the the request axis (iAxis - zero based) from the indicated portion of the coordinate system (pszTargetKey) which should be either "GEOGCS" or "PROJCS".

No CPLError is issued on routine failures (such as not finding the AXIS).

This method is equivalent to the C function **OSRGetAxis()** (p. ??).

**Parameters**

<i>pszTargetKey</i>	the coordinate system part to query ("PROJCS" or "GEOGCS").
<i>iAxis</i>	the axis to query (0 for first, 1 for second).
<i>peOrientation</i>	location into which to place the fetch orientation, may be NULL.

**Returns**

the name of the axis or NULL on failure.

References CPLDebug(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), and OGR\_SRSNode::GetValue().

**12.79.3.28** `const char * OGRSpatialReference::GetExtension ( const char * pszTargetKey, const char * pszName, const char * pszDefault = NULL ) const`

Fetch extension value.

Fetch the value of the named EXTENSION item for the identified target node.

#### Parameters

<i>pszTargetKey</i>	the name or path to the parent node of the EXTENSION.
<i>pszName</i>	the name of the extension being fetched.
<i>pszDefault</i>	the value to return if the extension is not found.

#### Returns

node value if successful or pszDefault on failure.

References OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), and OGR\_SRSNode::GetValue().

Referenced by exportToProj4().

**12.79.3.29** `double OGRSpatialReference::GetInvFlattening ( OGRErr * pnErr = NULL ) const`

Get spheroid inverse flattening.

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

#### Parameters

<i>pnErr</i>	if non-NULL set to OGRERR_FAILURE if no inverse flattening can be found.
--------------	--

#### Returns

inverse flattening, or SRS\_WGS84\_INVFLATTENING if it can't be found.

References CPLAtof(), GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), and OGR\_SRSNode::GetValue().

Referenced by exportToPanorama(), exportToPCI(), exportToProj4(), exportToUSGS(), GetSemiMinor(), and morphFromESRI().

**12.79.3.30** `double OGRSpatialReference::GetLinearUnits ( char ** ppszName = NULL ) const`

Fetch linear projection units.

If no units are available, a value of "Meters" and 1.0 will be assumed. This method only checks directly under the PROJCS, GEOCCS or LOCAL\_CS node for units.

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

#### Parameters

<i>ppszName</i>	a pointer to be updated with the pointer to the units name. The returned value remains internal to the <b>OGRSpatialReference</b> (p. ??) and shouldn't be freed, or modified. It may be invalidated on the next <b>OGRSpatialReference</b> (p. ??) call.
-----------------	---

#### Returns

the value to multiply by linear distances to transform them to meters.

References GetTargetLinearUnits().

Referenced by exportToERM(), exportToPCI(), exportToProj4(), importFromESRI(), importFromProj4(), IsSame(), morphToESRI(), SetLinearUnitsAndUpdateParameters(), and SetStatePlane().

**12.79.3.31** `double OGRSpatialReference::GetNormProjParm ( const char * pszName, double dfDefaultValue = 0.0, OGRErr * pnErr = NULL ) const`

Fetch a normalized projection parameter value.

This method is the same as **GetProjParm()** (p. ??) except that the value of the parameter is "normalized" into degrees or meters depending on whether it is linear or angular.

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

#### Parameters

<i>pszName</i>	the name of the parameter to fetch, from the set of SRS_PP codes in <b>ogr_srs_api.h</b> (p. ??).
<i>dfDefaultValue</i>	the value to return if this parameter doesn't exist.
<i>pnErr</i>	place to put error code on failure. Ignored if NULL.

#### Returns

value of parameter.

References **GetProjParm()**.

Referenced by **exportToPanorama()**, **exportToPCI()**, **exportToProj4()**, **exportToUSGS()**, **GetUTMZone()**, **morphToESRI()**, and **SetStatePlane()**.

**12.79.3.32** `double OGRSpatialReference::GetPrimeMeridian ( char ** ppszName = NULL ) const`

Fetch prime meridian info.

Returns the offset of the prime meridian from greenwich in degrees, and the prime meridian name (if requested). If no PRIMEM value exists in the coordinate system definition a value of "Greenwich" and an offset of 0.0 is assumed.

If the prime meridian name is returned, the pointer is to an internal copy of the name. It should not be freed, altered or depended on after the next OGR call.

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

#### Parameters

<i>ppszName</i>	return location for prime meridian name. If NULL, name is not returned.
-----------------	---

#### Returns

the offset to the GEOGCS prime meridian from greenwich in decimal degrees.

References **CPLAtof()**, **GetAttrNode()**, **OGR\_SRSNode::GetChild()**, **OGR\_SRSNode::GetChildCount()**, and **OGR\_SRSNode::GetValue()**.

Referenced by **morphFromESRI()**.

**12.79.3.33** `double OGRSpatialReference::GetProjParm ( const char * pszName, double dfDefaultValue = 0.0, OGRErr * pnErr = NULL ) const`

Fetch a projection parameter value.

NOTE: This code should be modified to translate non degree angles into degrees based on the GEOGCS unit. This has not yet been done.

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

## Parameters

<i>pszName</i>	the name of the parameter to fetch, from the set of SRS_PP codes in <b>ogr_srs_api.h</b> (p. ??).
<i>dfDefaultValue</i>	the value to return if this parameter doesn't exist.
<i>pnErr</i>	place to put error code on failure. Ignored if NULL.

## Returns

value of parameter.

References CPLAtof(), FindProjParm(), GetAttrNode(), OGR\_SRSNode::GetChild(), and OGR\_SRSNode::GetValue().

Referenced by GetNormProjParm(), GetUTMZone(), importFromProj4(), IsSame(), morphFromESRI(), morphToESRI(), and SetLinearUnitsAndUpdateParameters().

**12.79.3.34** `int OGRSpatialReference::GetReferenceCount ( ) const [inline]`

Fetch current reference count.

## Returns

the current reference count.

**12.79.3.35** `double OGRSpatialReference::GetSemiMajor ( OGRErr * pnErr = NULL ) const`

Get spheroid semi major axis.

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

## Parameters

<i>pnErr</i>	if non-NULL set to OGRERR_FAILURE if semi major axis can be found.
--------------	--

## Returns

semi-major axis, or SRS\_WGS84\_SEMIMAJOR if it can't be found.

References CPLAtof(), GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), and OGR\_SRSNode::GetValue().

Referenced by exportToPanorama(), exportToPCI(), exportToProj4(), exportToUSGS(), GetSemiMinor(), and morphFromESRI().

**12.79.3.36** `double OGRSpatialReference::GetSemiMinor ( OGRErr * pnErr = NULL ) const`

Get spheroid semi minor axis.

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

## Parameters

<i>pnErr</i>	if non-NULL set to OGRERR_FAILURE if semi minor axis can be found.
--------------	--

## Returns

semi-minor axis, or WGS84 semi minor if it can't be found.

References GetInvFlattening(), and GetSemiMajor().

Referenced by exportToProj4().

**12.79.3.37** `double OGRSpatialReference::GetTargetLinearUnits ( const char * pszTargetKey, char ** ppszName = NULL ) const`

Fetch linear units for target.

If no units are available, a value of "Meters" and 1.0 will be assumed.

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

#### Parameters

<i>pszTargetKey</i>	the key to look on. ie. "PROJCS" or "VERT_CS".
<i>ppszName</i>	a pointer to be updated with the pointer to the units name. The returned value remains internal to the <b>OGRSpatialReference</b> (p. ??) and shouldn't be freed, or modified. It may be invalidated on the next <b>OGRSpatialReference</b> (p. ??) call.

#### Returns

the value to multiply by linear distances to transform them to meters.

#### Since

OGR 1.9.0

References CPLAtof(), GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), OGR\_SRSNode::GetValue(), and IsVertical().

Referenced by GetLinearUnits().

**12.79.3.38** `OGRErr OGRSpatialReference::GetTOWGS84 ( double * padfCoeff, int nCoeffCount = 7 ) const`

Fetch TOWGS84 parameters, if available.

#### Parameters

<i>padfCoeff</i>	array into which up to 7 coefficients are placed.
<i>nCoeffCount</i>	size of padfCoeff - defaults to 7.

#### Returns

OGRErr\_NONE on success, or OGRErr\_FAILURE if there is no TOWGS84 node available.

References CPLAtof(), GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), and OGR\_SRSNode::GetValue().

Referenced by exportToPCI(), and IsSameGeogCS().

**12.79.3.39** `int OGRSpatialReference::GetUTMZone ( int * pbNorth = NULL ) const`

Get utm zone information.

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

In SWIG bindings (Python, Java, etc) the **GetUTMZone()** (p. ??) method returns a zone which is negative in the southern hemisphere instead of having the pbNorth flag used in the C and C++ interface.

#### Parameters

<i>pbNorth</i>	pointer to in to set to TRUE if northern hemisphere, or FALSE if southern.
----------------	--

#### Returns

UTM zone number or zero if this isn't a UTM definition.

References `GetAttrValue()`, `GetNormProjParm()`, and `GetProjParm()`.

Referenced by `AutoIdentifyEPSG()`, `exportToERM()`, `exportToPanorama()`, `exportToPCI()`, `exportToProj4()`, `exportToUSGS()`, and `morphToESRI()`.

#### 12.79.3.40 OGRErr OGRSpatialReference::importFromCRSURL ( const char \* *pszURL* )

Initialize from OGC URL.

Initializes this spatial reference from a coordinate system defined by an OGC URL prefixed with "http://opengis.net/def/crs" per best practice paper 11-135. Currently EPSG and OGC authority values are supported, including OGC auto codes, but not including CRS1 or CRS88 (NAVD88).

This method is also supported through `SetFromUserInput()` (p. ??) which can normally be used for URLs.

#### Parameters

<i>pszURL</i>	the URL string.
---------------	-----------------

#### Returns

OGRERR\_NONE on success or an error code.

References `OGR_SRSNode::AddChild()`, `Clear()`, `OGR_SRSNode::Clone()`, `CPLError()`, `CPLMalloc()`, `CPLStrdup()`, `OGR_SRSNode::GetValue()`, `importFromCRSURL()`, and `SetNode()`.

Referenced by `importFromCRSURL()`, and `SetFromUserInput()`.

#### 12.79.3.41 OGRErr OGRSpatialReference::importFromDict ( const char \* *pszDictFile*, const char \* *pszCode* )

Read SRS from WKT dictionary.

This method will attempt to find the indicated coordinate system identity in the indicated dictionary file. If found, the WKT representation is imported and used to initialize this **OGRSpatialReference** (p. ??).

More complete information on the format of the dictionary files can be found in the `epsg.wkt` file in the GDAL data tree. The dictionary files are searched for in the "GDAL" domain using `CPLFindFile()`. Normally this results in searching `/usr/local/share/gdal` or somewhere similar.

This method is the same as the C function `OSRImportFromDict()`.

#### Parameters

<i>pszDictFile</i>	the name of the dictionary file to load.
<i>pszCode</i>	the code to lookup in the dictionary.

#### Returns

OGRERR\_NONE on success, or OGRERR\_SRS\_UNSUPPORTED if the code isn't found, and OGRERR\_SRS\_FAILURE if something more dramatic goes wrong.

References `CPLReadLine()`, and `importFromWkt()`.

Referenced by `exportToERM()`, `importFromEPSGA()`, `importFromERM()`, and `SetFromUserInput()`.



**12.79.3.42 OGRErr OGRSpatialReference::importFromEPSG ( int *nCode* )**

Initialize SRS based on EPSG GCS or PCS code.

This method will initialize the spatial reference based on the passed in EPSG GCS or PCS code. The coordinate system definitions are normally read from the EPSG derived support files such as pcs.csv, gcs.csv, pcs.override.csv, gcs.override.csv and falling back to search for a PROJ.4 epsg init file or a definition in epsg.wkt.

These support files are normally searched for in /usr/local/share/gdal or in the directory identified by the GDAL\_DATA configuration option. See CPLFindFile() for details.

This method is relatively expensive, and generally involves quite a bit of text file scanning. Reasonable efforts should be made to avoid calling it many times for the same coordinate system.

This method is similar to **importFromEPSGA()** (p. ??) except that EPSG preferred axis ordering will *not* be applied for geographic coordinate systems. EPSG normally defines geographic coordinate systems to use lat/long contrary to typical GIS use). Since OGR 1.10.0, EPSG preferred axis ordering will also *not* be applied for projected coordinate systems that use northing/easting order.

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

**Parameters**

<i>nCode</i>	a GCS or PCS code from the horizontal coordinate system table.
--------------	--

**Returns**

OGRERR\_NONE on success, or an error code on failure.

References EPSGTreatsAsNorthingEasting(), GetAttrNode(), importFromEPSGA(), and OGR\_SRSNode::StripNodes().

Referenced by importFromERM(), importFromESRI(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), morphFromESRI(), SetFromUserInput(), SetStatePlane(), and SetWellKnownGeogCS().

**12.79.3.43 OGRErr OGRSpatialReference::importFromEPSGA ( int *nCode* )**

Initialize SRS based on EPSG GCS or PCS code.

This method will initialize the spatial reference based on the passed in EPSG GCS or PCS code.

This method is similar to **importFromEPSG()** (p. ??) except that EPSG preferred axis ordering *will* be applied for geographic and projected coordinate systems. EPSG normally defines geographic coordinate systems to use lat/long, and also there are also a few projected coordinate systems that use northing/easting order contrary to typical GIS use). See **OGRSpatialReference::importFromEPSG()** (p. ??) for more details on operation of this method.

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

**Parameters**

<i>nCode</i>	a GCS or PCS code from the horizontal coordinate system table.
--------------	--

**Returns**

OGRERR\_NONE on success, or an error code on failure.

References CPLError(), FixupOrdering(), GetAuthorityName(), importFromDict(), importFromProj4(), IsGeographic(), IsProjected(), and SetAuthority().

Referenced by importFromEPSG(), SetFromUserInput(), and SetWellKnownGeogCS().

#### 12.79.3.44 OGRErr OGRSpatialReference::importFromERM ( const char \* *pszProj*, const char \* *pszDatum*, const char \* *pszUnits* )

Create OGR WKT from ERMapper projection definitions.

Generates an **OGRSpatialReference** (p. ??) definition from an ERMapper datum and projection name. Based on the ecw\_cs.wkt dictionary file from gdal/data.

##### Parameters

<i>pszProj</i>	the projection name, such as "NUTM11" or "GEOGRAPHIC".
<i>pszDatum</i>	the datum name, such as "NAD83".
<i>pszUnits</i>	the linear units "FEET" or "METERS".

##### Returns

OGRErr\_NONE on success or OGRErr\_UNSUPPORTED\_SRS if not found.

References Clear(), CopyGeogCSFrom(), importFromDict(), importFromEPSG(), IsLocal(), and SetLinearUnits().

#### 12.79.3.45 OGRErr OGRSpatialReference::importFromESRI ( char \*\* *papszPrj* )

Import coordinate system from ESRI .prj format(s).

This function will read the text loaded from an ESRI .prj file, and translate it into an **OGRSpatialReference** (p. ??) definition. This should support many (but by no means all) old style (Arc/Info 7.x) .prj files, as well as the newer pseudo-OGC WKT .prj files. Note that new style .prj files are in OGC WKT format, but require some manipulation to correct datum names, and units on some projection parameters. This is addressed within **importFromESRI()** (p. ??) by an automatical call to **morphFromESRI()** (p. ??).

Currently only GEOGRAPHIC, UTM, STATEPLANE, GREATBRITIAN\_GRID, ALBERS, EQUIDISTANT\_CONIC, TRANSVERSE (mercator), POLAR, MERCATOR and POLYCONIC projections are supported from old style files.

At this time there is no equivalent exportToESRI() method. Writing old style .prj files is not supported by **OGRSpatialReference** (p. ??). However the **morphToESRI()** (p. ??) and **exportToWkt()** (p. ??) methods can be used to generate output suitable to write to new style (Arc 8) .prj files.

This function is the equivalent of the C function **OSRImportFromESRI()** (p. ??).

##### Parameters

<i>papszPrj</i>	NULL terminated list of strings containing the definition.
-----------------	--

##### Returns

OGRErr\_NONE on success or an error code in case of failure.

References CopyGeogCSFrom(), CPLDebug(), CPLRealloc(), CPLStrdup(), OGR\_SRSNode::DestroyChild(), GetAttrNode(), GetLinearUnits(), importFromEPSG(), importFromWkt(), IsLocal(), IsProjected(), morphFromESRI(), SetACEA(), SetEC(), SetLAEA(), SetLCC(), SetLinearUnitsAndUpdateParameters(), SetLocalCS(), SetMercator(), SetPolyconic(), SetPS(), SetStatePlane(), SetTM(), SetUTM(), and SetWellKnownGeogCS().

#### 12.79.3.46 OGRErr OGRSpatialReference::importFromMICoordSys ( const char \* *pszCoordSys* )

Import Mapinfo style CoordSys definition.

The **OGRSpatialReference** (p. ??) is initialized from the passed Mapinfo style CoordSys definition string.

This method is the equivalent of the C function **OSRImportFromMICoordSys()** (p. ??).

## Parameters

<i>pszCoordSys</i>	Mapinfo style CoordSys definition string.
--------------------	---

## Returns

OGRERR\_NONE on success, OGRERR\_FAILURE on failure, OGRERR\_UNSUPPORTED\_OPERATION if MITAB library was not linked in.

References CPLError().

#### 12.79.3.47 OGRErr OGRSpatialReference::importFromOzi ( const char \* *pszDatum*, const char \* *pszProj*, const char \* *pszProjParms* )

Note : This method is obsolete, but has been kept to avoid breaking the API. It can be removed in GDAL 2.0 Import coordinate system from OziExplorer projection definition.

This method will import projection definition in style, used by OziExplorer software.

This function is the equivalent of the C function OSRImportFromOzi().

## Parameters

<i>pszDatum</i>	Datum string. This is a fifth string in the OziExplorer .MAP file.
<i>pszProj</i>	Projection string. Search for line starting with "Map Projection" name in the OziExplorer .MAP file and supply it as a whole in this parameter.
<i>pszProjParms</i>	String containing projection parameters. Search for "Projection Setup" name in the OziExplorer .MAP file and supply it as a whole in this parameter.

## Returns

OGRERR\_NONE on success or an error code in case of failure.

**Deprecated** Use `importFromOzi( const char * const* papszLines )` (p. ??) instead

#### 12.79.3.48 OGRErr OGRSpatialReference::importFromOzi ( const char \*const \* *papszLines* )

Import coordinate system from OziExplorer projection definition.

This method will import projection definition in style, used by OziExplorer software.

## Parameters

<i>papszLines</i>	Map file lines. This is an array of strings containing the whole OziExplorer .MAP file. The array is terminated by a NULL pointer.
-------------------	--

## Returns

OGRERR\_NONE on success or an error code in case of failure.

## Since

OGR 1.10

References `Clear()`, `CopyGeogCSFrom()`, `CPLAtof()`, `CPLAtofM()`, `CPLDebug()`, `CPLError()`, `CSLCount()`, `CSLDestroy()`, `CSLTokenizeString2()`, `FixupOrdering()`, `importFromEPSG()`, `IsLocal()`, `IsProjected()`, `SetACEA()`, `SetGeogCS()`, `SetLCC()`, `SetLCC1SP()`, `SetLinearUnits()`, `SetLocalCS()`, `SetMercator()`, `SetSinusoidal()`, `SetTM()`, `SetTOWGS84()`, and `SetUTM()`.

### 12.79.3.49 OGRErr OGRSpatialReference::importFromPanorama ( long *iProjSys*, long *iDatum*, long *iEllips*, double \* *padfPrjParams* )

Import coordinate system from "Panorama" GIS projection definition.

This method will import projection definition in style, used by "Panorama" GIS.

This function is the equivalent of the C function OSRImportFromPanorama().

#### Parameters

<i>iProjSys</i>	<p>Input projection system code, used in GIS "Panorama".</p> <p>&lt;h4&gt;Supported Projections&lt;/h4&gt;</p> <ul style="list-style-type: none"> <li>1: Gauss-Kruger (Transverse Mercator)</li> <li>2: Lambert Conformal Conic 2SP</li> <li>5: Stereographic</li> <li>6: Azimuthal Equidistant (Postel)</li> <li>8: Mercator</li> <li>10: Polyconic</li> <li>13: Polar Stereographic</li> <li>15: Gnomonic</li> <li>17: Universal Transverse Mercator (UTM)</li> <li>18: Wagner I (Kavraisky VI)</li> <li>19: Mollweide</li> <li>20: Equidistant Conic</li> <li>24: Lambert Azimuthal Equal Area</li> <li>27: Equirectangular</li> <li>28: Cylindrical Equal Area (Lambert)</li> <li>29: International Map of the World Polyconic</li> </ul>
<i>iDatum</i>	<p>Input coordinate system.</p> <p>&lt;h4&gt;Supported Datums&lt;/h4&gt;</p> <ul style="list-style-type: none"> <li>1: Pulkovo, 1942</li> <li>2: WGS, 1984</li> <li>3: OSGB 1936 (British National Grid)</li> <li>9: Pulkovo, 1995</li> </ul>
<i>iEllips</i>	<p>Input spheroid.</p> <p>&lt;h4&gt;Supported Spheroids&lt;/h4&gt;</p> <ul style="list-style-type: none"> <li>1: Krassovsky, 1940</li> <li>2: WGS, 1972</li> <li>3: International, 1924 (Hayford, 1909)</li> <li>4: Clarke, 1880</li> <li>5: Clarke, 1866 (NAD1927)</li> <li>6: Everest, 1830</li> <li>7: Bessel, 1841</li> <li>8: Airy, 1830</li> <li>9: WGS, 1984 (GPS)</li> </ul>

<i>padfPrjParams</i>	Array of 8 coordinate system parameters:
----------------------	--

```
[0] Latitude of the first standard parallel (radians)
[1] Latitude of the second standard parallel (radians)
[2] Latitude of center of projection (radians)
[3] Longitude of center of projection (radians)
[4] Scaling factor
[5] False Easting
[6] False Northing
[7] Zone number
```

Particular projection uses different parameters, unused ones may be set to zero. If NULL supplied instead of array pointer default values will be used (i.e., zeroes).

#### Returns

OGRERR\_NONE on success or an error code in case of failure.

References Clear(), CopyGeogCSFrom(), CPLDebug(), CPLError(), CPLMalloc(), FixupOrdering(), importFromEPSG(), IsLocal(), IsProjected(), SetAE(), SetAuthority(), SetCEA(), SetEC(), SetEquirectangular(), SetGeogCS(), SetGnomonic(), SetIWMPolyconic(), SetLAEA(), SetLCC(), SetLinearUnits(), SetLocalCS(), SetMercator(), SetMollweide(), SetPolyconic(), SetPS(), SetStereographic(), SetTM(), SetUTM(), SetWagner(), and SetWellKnownGeogCS().

**12.79.3.50 OGRErr OGRSpatialReference::importFromPCI ( const char \* *pszProj*, const char \* *pszUnits* = NULL, double \* *padfPrjParams* = NULL )**

Import coordinate system from PCI projection definition.

PCI software uses 16-character string to specify coordinate system and datum/ellipsoid. You should supply at least this string to the **importFromPCI()** (p. ??) function.

This function is the equivalent of the C function **OSRIImportFromPCI()** (p. ??).

#### Parameters

<i>pszProj</i>	NULL terminated string containing the definition. Looks like "pppppppppppp Ennn" or "pppppppppppp Dnnn", where "pppppppppppp" is a projection code, "Ennn" is an ellipsoid code, "Dnnn" — a datum code.
<i>pszUnits</i>	Grid units code ("DEGREE" or "METRE"). If NULL "METRE" will be used.
<i>padfPrjParams</i>	Array of 17 coordinate system parameters:

[0] Spheroid semi major axis [1] Spheroid semi minor axis [2] Reference Longitude [3] Reference Latitude [4] First Standard Parallel [5] Second Standard Parallel [6] False Easting [7] False Northing [8] Scale Factor [9] Height above sphere surface [10] Longitude of 1st point on center line [11] Latitude of 1st point on center line [12] Longitude of 2nd point on center line [13] Latitude of 2nd point on center line [14] Azimuth east of north for center line [15] Landsat satellite number [16] Landsat path number

Particular projection uses different parameters, unused ones may be set to zero. If NULL supplied instead of array pointer default values will be used (i.e., zeroes).

#### Returns

OGRERR\_NONE on success or an error code in case of failure.

References Clear(), CopyGeogCSFrom(), CPLAtof(), CPLDebug(), CPLMalloc(), CPLScanLong(), CPLStrnlen(), CSLCount(), CSLDestroy(), FixupOrdering(), importFromEPSG(), IsGeographic(), IsLocal(), IsProjected(), SetACEA(), SetAE(), SetAngularUnits(), SetAuthority(), SetCS(), SetEC(), SetEquirectangular2(), SetGeogCS(), SetGnomonic(), SetHOM(), SetHOM2PNO(), SetLAEA(), SetLCC(), SetLCC1SP(), SetLinearUnits(), SetLinearUnitsAndUpdateParameters(), SetLocalCS(), SetMC(), SetMercator(), SetOrthographic(), SetOS(), SetPolyconic(), SetPS(), SetRobinson(), SetSinusoidal(), SetStatePlane(), SetStereographic(), SetTM(), SetTOWGS84(), SetUTM(), and SetVDG().

### 12.79.3.51 OGRErr OGRSpatialReference::importFromProj4 ( const char \* pszProj4 )

Import PROJ.4 coordinate string.

The **OGRSpatialReference** (p. ??) is initialized from the passed PROJ.4 style coordinate system string. In addition to many +proj formulations which have OGC equivalents, it is also possible to import "+init=epsg:n" style definitions. These are passed to **importFromEPSG()** (p. ??). Other init strings (such as the state plane zones) are not currently supported.

Example: pszProj4 = "+proj=utm +zone=11 +datum=WGS84"

Some parameters, such as grids, recognised by PROJ.4 may not be well understood and translated into the **OGRSpatialReference** (p. ??) model. It is possible to add the +wktext parameter which is a special keyword that OGR recognises as meaning "embed the entire PROJ.4 string in the WKT and use it literally when converting back to PROJ.4 format".

For example: "+proj=nzmg +lat\_0=-41 +lon\_0=173 +x\_0=2510000 +y\_0=6023150 +ellps=intl +units=m +nadgrids=nzgd2kgrid0005.gsb +wktext"

will be translated as :

```
* PROJCS["unnamed",
*   GEOGCS["International 1909 (Hayford)",
*     DATUM["unknown",
*       SPHEROID["intl",6378388,297]],
*     PRIMEM["Greenwich",0],
*     UNIT["degree",0.0174532925199433]],
*   PROJECTION["New_Zealand_Map_Grid"],
*   PARAMETER["latitude_of_origin",-41],
*   PARAMETER["central_meridian",173],
*   PARAMETER["false_easting",2510000],
*   PARAMETER["false_northing",6023150],
*   UNIT["Meter",1],
*   EXTENSION["PROJ4","+proj=nzmg +lat_0=-41 +lon_0=173 +x_0=2510000
*     +y_0=6023150 +ellps=intl +units=m +nadgrids=nzgd2kgrid0005.gsb +wktext"]]
*
```

Special processing for 'etmerc' (GDAL >= 1.10 ): if +proj=etmerc is found in the passed string, the SRS built will use the WKT representation for a standard Transverse Mercator, but will also include a PROJ4 EXTENSION node to preserve the etmerc projection method.

For example: "+proj=etmerc +lat\_0=0 +lon\_0=9 +k=0.9996 +units=m +x\_0=500000 +datum=WGS84"

will be translated as :

```
* PROJCS["unnamed",
*   GEOGCS["WGS 84",
*     DATUM["WGS_1984",
*       SPHEROID["WGS 84",6378137,298.257223563,
*         AUTHORITY["EPSG","7030"]],
*       TOWGS84[0,0,0,0,0,0,0],
*       AUTHORITY["EPSG","6326"]],
*     PRIMEM["Greenwich",0],
*     AUTHORITY["EPSG","8901"]],
*     UNIT["degree",0.0174532925199433,
*       AUTHORITY["EPSG","9108"]],
*     AUTHORITY["EPSG","4326"]],
*   PROJECTION["Transverse_Mercator"],
*   PARAMETER["latitude_of_origin",0],
*   PARAMETER["central_meridian",9],
*   PARAMETER["scale_factor",0.9996],
*   PARAMETER["false_easting",500000],
*   PARAMETER["false_northing",0],
*   UNIT["Meter",1],
*   EXTENSION["PROJ4","+proj=etmerc +lat_0=0 +lon_0=9 +k=0.9996 +units=m +x_0=500000 +datum=WGS84 +nodels
*     ]]
*
```

This method is the equivalent of the C function **OSRImportFromProj4()** (p. ??).

## Parameters

<i>pszProj4</i>	the PROJ.4 style string.
-----------------	--------------------------

## Returns

OGRERR\_NONE on success or OGRERR\_CORRUPT\_DATA on failure.

References OGR\_SRSNode::AddChild(), Clear(), OGR\_SRSNode::Clone(), CopyGeogCSFrom(), CPLAtof(), CPLAtofM(), CPLDebug(), CPLError(), CPLStrdup(), CSLCount(), CSLDestroy(), GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), GetLinearUnits(), OGR\_SRSNode::GetNode(), GetProjParm(), OGR\_SRSNode::GetValue(), importFromEPSG(), IsGeocentric(), IsLocal(), IsProjected(), SetACEA(), SetAE(), SetBonne(), SetCEA(), SetCS(), SetEC(), SetEckert(), SetEquirectangular(), SetEquirectangular2(), SetExtension(), SetGaussSchreiberTMercator(), SetGeocCS(), SetGeogCS(), SetGEOS(), SetGH(), SetGnomonic(), SetGS(), SetHOM(), SetHOMAC(), SetIGH(), SetIWMPolyconic(), SetKrovak(), SetLAEA(), SetLCC(), SetLCC1SP(), SetLinearUnits(), SetMC(), SetMercator(), SetMollweide(), SetNode(), SetNormProjParm(), SetNZMG(), SetOrthographic(), SetOS(), SetPolyconic(), SetProjection(), SetPS(), SetRobinson(), SetSinusoidal(), SetStereographic(), SetTM(), SetTMSO(), SetTOWGS84(), SetTPED(), SetUTM(), SetVDG(), SetWagner(), and SetWellKnownGeogCS().

Referenced by importFromEPSGA(), and SetFromUserInput().

### 12.79.3.52 OGRErr OGRSpatialReference::importFromUrl ( const char \* *pszUrl* )

Set spatial reference from a URL.

This method will download the spatial reference at a given URL and feed it into SetFromUserInput for you.

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

## Parameters

<i>pszUrl</i>	text definition to try to deduce SRS from.
---------------	--

## Returns

OGRERR\_NONE on success, or an error code with the curl error message if it is unable to download data.

References CPLError(), CPLErrorReset(), CPLGetLastErrorNo(), CPLHTTPDestroyResult(), CPLHTTPFetch(), CPLHTTPResult::nDataLen, CPLHTTPResult::nStatus, CPLHTTPResult::pabyData, CPLHTTPResult::pszErrBuf, and SetFromUserInput().

Referenced by SetFromUserInput().

### 12.79.3.53 OGRErr OGRSpatialReference::importFromURN ( const char \* *pszURN* )

Initialize from OGC URN.

Initializes this spatial reference from a coordinate system defined by an OGC URN prefixed with "urn:ogc:def:crs:" per recommendation paper 06-023r1. Currently EPSG and OGC authority values are supported, including OGC auto codes, but not including CRS1 or CRS88 (NAVD88).

This method is also support through **SetFromUserInput()** (p. ??) which can normally be used for URNs.

## Parameters

<i>pszURN</i>	the urn string.
---------------	-----------------

## Returns

OGRERR\_NONE on success or an error code.

References OGR\_SRSNode::AddChild(), Clear(), OGR\_SRSNode::Clone(), CPLError(), CPLStrdup(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetValue(), and SetNode().

Referenced by SetFromUserInput().

12.79.3.54 OGRErr OGRSpatialReference::importFromUSGS ( long *iProjSys*, long *iZone*, double \* *padfPrjParams*, long *iDatum*, int *nUSGSAngleFormat* = TRUE )

Import coordinate system from USGS projection definition.

This method will import projection definition in style, used by USGS GCTP software. GCTP operates on angles in packed DMS format (see **CPLDecToPackedDMS()** (p. ??) function for details), so all angle values (latitudes, longitudes, azimuths, etc.) specified in the *padfPrjParams* array should be in the packed DMS format, unless *bAnglesInPackedDMSFormat* is set to FALSE.

This function is the equivalent of the C function **OSRImportFromUSGS()** (p. ??). Note that the *bAnglesInPackedDMSFormat* parameter is only present in the C++ method. The C function assumes *bAnglesInPackedFormat* = TRUE.



## Parameters

<i>iProjSys</i>	Input projection system code, used in GCTP.									
<i>iZone</i>	Input zone for UTM and State Plane projection systems. For Southern Hemisphere UTM use a negative zone code. iZone ignored for all other projections.									
<i>padfPrjParams</i>	Array of 15 coordinate system parameters. These parameters differs for different projections.									
<div>&lt;h4&gt;Projection Transformation Package Projection Parameters&lt;/h4&gt;</div>										
<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div>&lt;</div>										

```

----- | Array Element | Code & Projection Id |----- | 8 | 9 | 10 |
11 | 12 | ----- 0 Geographic | | | | 1 U T M | | | | 2 State Plane | | | | 3
Albers Equal Area | | | | 4 Lambert Conformal C | | | | 5 Mercator | | | | 6 Polar Stereographic | | | | 7
Polyconic | | | | 8 Equid. Conic A |zero | | | | Equid. Conic B |one | | | | 9 Transverse Mercator | | | | 10
Stereographic | | | | 11 Lambert Azimuthal | | | | 12 Azimuthal | | | | 13 Gnomonic | | | | 14 Orthographic
| | | | 15 Gen. Vert. Near Per | | | | 16 Sinusoidal | | | | 17 Equirectangular | | | | 18 Miller Cylindrical | |
| | | 19 Van der Grinten | | | | 20 Hotin Oblique Merc A |Long1|Lat1|Long2|Lat2|zero| Hotin Oblique Merc B |
| | |one | 21 Robinson | | | | 22 Space Oblique Merc A |PSRev|LRat|PFlag |zero| Space Oblique Merc B | |
| | |one | 23 Alaska Conformal | | | | 24 Interrupted Goode | | | | 25 Mollweide | | | | 26 Interrupt Mollweide
| | | | 27 Hammer | | | | 28 Wagner IV | | | | 29 Wagner VII | | | | 30 Oblated Equal Area |Angle| | | |
-----

```

where

Lon/Z	Longitude of any point in the UTM zone or zero. If zero, a zone code must be specified.
Lat/Z	Latitude of any point in the UTM zone or zero. If zero, a zone code must be specified.
SMajor	Semi-major axis of ellipsoid. If zero, Clarke 1866 in meters is assumed.
SMinor	Eccentricity squared of the ellipsoid if less than zero, if zero, a spherical form is assumed, or if greater than zero, the semi-minor axis of ellipsoid.
Sphere	Radius of reference sphere. If zero, 6370997 meters is used.
STDPAR	Latitude of the standard parallel
STDPR1	Latitude of the first standard parallel
STDPR2	Latitude of the second standard parallel
CentMer	Longitude of the central meridian
OriginLat	Latitude of the projection origin
FE	False easting in the same units as the semi-major axis
FN	False northing in the same units as the semi-major axis
TrueScale	Latitude of true scale
LongPol	Longitude down below pole of map
Factor	Scale factor at central meridian (Transverse Mercator) or center of projection (Hotine Oblique Mercator)
CentLon	Longitude of center of projection
CenterLat	Latitude of center of projection
Height	Height of perspective point
Long1	Longitude of first point on center line (Hotine Oblique Mercator, format A)
Long2	Longitude of second point on center line (Hotine Oblique Mercator, format A)
Lat1	Latitude of first point on center line (Hotine Oblique Mercator, format A)
Lat2	Latitude of second point on center line (Hotine Oblique Mercator, format A)
AziAng	Azimuth angle east of north of center line (Hotine Oblique Mercator, format B)
AzmthPt	Longitude of point on central meridian where azimuth occurs (Hotine Oblique Mercator, format B)
IncAng	Inclination of orbit at ascending node, counter-clockwise from equator (SOM, format A)
AscLong	Longitude of ascending orbit at equator (SOM, format A)
PSRev	Period of satellite revolution in minutes (SOM, format A)
LRat	Landsat ratio to compensate for confusion at northern end of orbit (SOM, format A -- use 0.5201613)
PFlag	End of path flag for Landsat: 0 = start of path, 1 = end of path (SOM, format A)
Satnum	Landsat Satellite Number (SOM, format B)
Path	Landsat Path Number (Use WRS-1 for Landsat 1, 2 and 3 and WRS-2 for Landsat 4, 5 and 6.) (SOM, format B)
Shapem	Oblated Equal Area oval shape parameter m
Shapen	Oblated Equal Area oval shape parameter n
Angle	Oblated Equal Area oval rotation angle

Array elements 13 and 14 are set to zero. All array elements with blank fields are set to zero too.

#### Parameters

<i>iDatum</i>	Input spheroid.
---------------	-----------------

If the datum code is negative, the first two values in the parameter array (parm) are used to define the values as follows:

- If `padfPrjParams[0]` is a non-zero value and `padfPrjParams[1]` is greater than one, the semimajor axis is set to `padfPrjParams[0]` and the semiminor axis is set to `padfPrjParams[1]`.

- If `padfPrjParams[0]` is nonzero and `padfPrjParams[1]` is greater than zero but less than or equal to one, the semimajor axis is set to `padfPrjParams[0]` and the semiminor axis is computed from the eccentricity squared value `padfPrjParams[1]`:

$$\text{semiminor} = \sqrt{1.0 - \text{ES}} * \text{semimajor}$$

where

ES = eccentricity squared

- If `padfPrjParams[0]` is nonzero and `padfPrjParams[1]` is equal to zero, the semimajor axis and semiminor axis are set to `padfPrjParams[0]`.
- If `padfPrjParams[0]` equals zero and `padfPrjParams[1]` is greater than zero, the default Clarke 1866 is used to assign values to the semimajor axis and semiminor axis.
- If `padfPrjParams[0]` and `padfPrjParams[1]` equals zero, the semimajor axis is set to 6370997.0 and the semiminor axis is set to zero.

If a datum code is zero or greater, the semimajor and semiminor axis are defined by the datum code as found in the following table:

<h4>Supported Datums</h4>

```

0: Clarke 1866 (default)
1: Clarke 1880
2: Bessel
3: International 1967
4: International 1909
5: WGS 72
6: Everest
7: WGS 66
8: GRS 1980/WGS 84
9: Airy
10: Modified Everest
11: Modified Airy
12: Walbeck
13: Southeast Asia
14: Australian National
15: Krassovsky
16: Hough
17: Mercury 1960
18: Modified Mercury 1968
19: Sphere of Radius 6370997 meters

```

#### Parameters

<i>nUSGSAngle-Format</i>	one of USGS_ANGLE_DECIMALDEGREES, USGS_ANGLE_PACKEDDMS, or USGS_ANGLE_RADIANS (default is USGS_ANGLE_PACKEDDMS).
--------------------------	--

#### Returns

OGRERR\_NONE on success or an error code in case of failure.

References CPLDebug(), CPLError(), CPLPackedDMSToDec(), FixupOrdering(), IsLocal(), IsProjected(), SetAC-EA(), SetAE(), SetAuthority(), SetEC(), SetEquiangular2(), SetGeogCS(), SetGnomonic(), SetHOM(), SetHOM2PNO(), SetLAEA(), SetLCC(), SetLinearUnits(), SetLocalCS(), SetMC(), SetMercator(), SetMollweide(), SetOrthographic(), SetPolyconic(), SetPS(), SetRobinson(), SetSinusoidal(), SetStatePlane(), SetStereographic(), SetTM(), SetUTM(), SetVDG(), SetWagner(), and SetWellKnownGeogCS().

### 12.79.3.55 OGRErr OGRSpatialReference::importFromWkt ( char \*\* *ppszInput* )

Import from WKT string.

This method will wipe the existing SRS definition, and reassign it based on the contents of the passed WKT string. Only as much of the input string as needed to construct this SRS is consumed from the input string, and the input string pointer is then updated to point to the remaining (unused) input.

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

#### Parameters

<i>ppszInput</i>	Pointer to pointer to input. The pointer is updated to point to remaining unused input text.
------------------	--

#### Returns

OGRERR\_NONE if import succeeds, or OGRERR\_CORRUPT\_DATA if it fails for any reason.

References OGR\_SRSNode::AddChild(), and Clear().

Referenced by importFromDict(), importFromESRI(), OGRSpatialReference(), OSRNewSpatialReference(), SetFromUserInput(), and SetWellKnownGeogCS().

### 12.79.3.56 OGRErr OGRSpatialReference::importFromWMSAUTO ( const char \* *pszDefinition* )

Initialize from WMSAUTO string.

Note that the WMS 1.3 specification does not include the units code, while apparently earlier specs do. We try to guess around this.

#### Parameters

<i>pszDefinition</i>	the WMSAUTO string
----------------------	--------------------

#### Returns

OGRERR\_NONE on success or an error code.

References Clear(), CPLAtof(), CPLError(), CSLCount(), CSLDestroy(), SetAuthority(), SetEquirectangular(), SetLinearUnits(), SetMollweide(), SetOrthographic(), SetTM(), SetUTM(), and SetWellKnownGeogCS().

Referenced by SetFromUserInput().

### 12.79.3.57 OGRErr OGRSpatialReference::importFromXML ( const char \* *pszXML* )

Import coordinate system from XML format (GML only currently).

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

#### Parameters

<i>pszXML</i>	XML string to import
---------------	----------------------

#### Returns

OGRERR\_NONE on success or OGRERR\_CORRUPT\_DATA on failure.

References Clear(), CPLDestroyXMLNode(), CPLParseXMLString(), CPLStripXMLNamespace(), CPLXMLNode::psNext, and CPLXMLNode::pszValue.

Referenced by SetFromUserInput().

**12.79.3.58 int OGRSpatialReference::IsCompound ( ) const**

Check if coordinate system is compound.

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

**Returns**

TRUE if this is rooted with a COMPD\_CS node.

References OGR\_SRSNode::GetValue().

**12.79.3.59 int OGRSpatialReference::IsGeocentric ( ) const**

Check if geocentric coordinate system.

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

**Returns**

TRUE if this contains a GEOCCS node indicating a it is a geocentric coordinate system.

**Since**

OGR 1.9.0

References OGR\_SRSNode::GetValue().

Referenced by CloneGeogCS(), CopyGeogCSFrom(), exportToProj4(), importFromProj4(), and SetGeogCS().

**12.79.3.60 int OGRSpatialReference::IsGeographic ( ) const**

Check if geographic coordinate system.

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

**Returns**

TRUE if this spatial reference is geographic ... that is the root is a GEOGCS node.

References GetAttrNode(), and OGR\_SRSNode::GetValue().

Referenced by AutoidentifyEPSG(), EPSGTreatsAsLatLong(), exportToERM(), exportToProj4(), exportToXML(), importFromEPSGA(), importFromPCI(), SetCompoundCS(), SetVertCS(), and SetWellKnownGeogCS().

**12.79.3.61 int OGRSpatialReference::IsLocal ( ) const**

Check if local coordinate system.

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

**Returns**

TRUE if this spatial reference is local ... that is the root is a LOCAL\_CS node.

Referenced by exportToPanorama(), exportToPCI(), exportToUSGS(), importFromERM(), importFromESRI(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), importFromUSGS(), and IsSame().

### 12.79.3.62 int OGRSpatialReference::IsProjected ( ) const

Check if projected coordinate system.

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

#### Returns

TRUE if this contains a PROJCS node indicating a it is a projected coordinate system.

References GetAttrNode(), and OGR\_SRSNode::GetValue().

Referenced by AutoidentifyEPSG(), EPSGTreatsAsNorthingEasting(), exportToERM(), exportToXML(), importFromEPSGA(), importFromESRI(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), importFromUSGS(), IsSame(), SetCompoundCS(), and SetVertCS().

### 12.79.3.63 int OGRSpatialReference::IsSame ( const OGRSpatialReference \* *poOtherSRS* ) const

Do these two spatial references describe the same system ?

#### Parameters

<i>poOtherSRS</i>	the SRS being compared to.
-------------------	----------------------------

#### Returns

TRUE if equivalent or FALSE otherwise.

References GetAttrNode(), GetAttrValue(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), GetLinearUnits(), GetProjParm(), OGR\_SRSNode::GetValue(), IsLocal(), IsProjected(), IsSameGeogCS(), IsSameVertCS(), and IsVertical().

Referenced by OGRGeometry::Difference(), OGRGeometry::Intersection(), OGRGeometry::SymDifference(), and OGRGeometry::Union().

### 12.79.3.64 int OGRSpatialReference::IsSameGeogCS ( const OGRSpatialReference \* *poOther* ) const

Do the GeogCS'es match?

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

#### Parameters

<i>poOther</i>	the SRS being compared against.
----------------	---------------------------------

#### Returns

TRUE if they are the same or FALSE otherwise.

References CPLAtof(), GetAttrValue(), and GetTOWGS84().

Referenced by IsSame(), and morphFromESRI().

### 12.79.3.65 int OGRSpatialReference::IsSameVertCS ( const OGRSpatialReference \* *poOther* ) const

Do the VertCS'es match?

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

## Parameters

<i>poOther</i>	the SRS being compared against.
----------------	---------------------------------

## Returns

TRUE if they are the same or FALSE otherwise.

References CPLAtof(), and GetAttrValue().

Referenced by IsSame().

## 12.79.3.66 int OGRSpatialReference::IsVertical ( ) const

Check if vertical coordinate system.

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

## Returns

TRUE if this contains a VERT\_CS node indicating a it is a vertical coordinate system.

## Since

OGR 1.8.0

References GetAttrNode(), and OGR\_SRSNode::GetValue().

Referenced by GetTargetLinearUnits(), IsSame(), SetCompoundCS(), and SetTargetLinearUnits().

## 12.79.3.67 OGRErr OGRSpatialReference::morphFromESRI ( )

Convert in place from ESRI WKT format.

The value notes of this coordinate system are modified in various manners to adhere more closely to the WKT standard. This mostly involves translating a variety of ESRI names for projections, arguments and datums to "standard" names, as defined by Adam Gawne-Cain's reference translation of EPSG to WKT for the CT specification.

Starting with GDAL 1.9.0, missing parameters in TOWGS84, DATUM or GEOGCS nodes can be added to the WKT, comparing existing WKT parameters to GDAL's databases. Note that this optional procedure is very conservative and should not introduce false information into the WKT definition (although caution should be advised when activating it). Needs the Configuration Option GDAL\_FIX\_ESRI\_WKT be set to one of the following values (TOWGS84 is recommended for proper datum shift calculations):

## GDAL\_FIX\_ESRI\_WKT values

	<b>TOWGS84</b>		Adds missing TOWGS84 parameters (necessary for datum transformations), based on named datum and spheroid values.
--	----------------	--	--

	<b>DATUM</b>		Adds EPSG AUTHORITY nodes and sets SPHEROID name to OGR spec.
	<b>GEOGCS</b>		Adds EPSG AUTHORITY nodes and sets GEOGCS, DATUM and SPHEROID names to OGR spec. Effectively replaces GEOGCS node with the result of importFromEPSG(n), using EPSG code n corresponding to the existing GEOGCS. Does not impact PROJCS values.

This does the same as the C function **OSRMorphFromESRI()** (p. ??).

#### Returns

OGRERR\_NONE unless something goes badly wrong.

References OGR\_SRSNode::AddChild(), OGR\_SRSNode::applyRemapper(), OGR\_SRSNode::Clone(), CloneGeogCS(), CopyGeogCSFrom(), CPLDebug(), CPLGetConfigOption(), CPLStrdup(), OGR\_SRSNode::DestroyChild(), exportToWkt(), OGR\_SRSNode::FindChild(), FixupOrdering(), GetAttrNode(), GetAttrValue(), OGR\_SRSNode::GetChild(), GetInvFlattening(), GetPrimeMeridian(), GetProjParm(), GetSemiMajor(), OGR\_SRSNode::GetValue(), importFromEPSG(), OGR\_SRSNode::InsertChild(), IsSameGeogCS(), SetNode(), SetProjParm(), OGR\_SRSNode::SetValue(), and StripCTParms().

Referenced by importFromESRI(), and SetFromUserInput().

#### 12.79.3.68 OGRErr OGRSpatialReference::morphToESRI ( )

Convert in place to ESRI WKT format.

The value nodes of this coordinate system are modified in various manners more closely map onto the ESRI concept of WKT format. This includes renaming a variety of projections and arguments, and stripping out nodes not recognised by ESRI (like AUTHORITY and AXIS).

This does the same as the C function **OSRMorphToESRI()** (p. ??).

#### Returns

OGRERR\_NONE unless something goes badly wrong.

References OGR\_SRSNode::AddChild(), OGR\_SRSNode::applyRemapper(), CPLDebug(), CPLMalloc(), CPLStrdup(), OGR\_SRSNode::DestroyChild(), FindProjParm(), Fixup(), GetAngularUnits(), GetAttrNode(), GetAttrValue(), GetAuthorityCode(), GetAuthorityName(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), GetLinearUnits(), GetNormProjParm(), GetProjParm(), GetUTMZone(), OGR\_SRSNode::GetValue(), SetNode(), OGR\_SRSNode::SetValue(), and StripCTParms().

#### 12.79.3.69 int OGRSpatialReference::Reference ( )

Increments the reference count by one.

The reference count is used keep track of the number of **OGRGeometry** (p. ??) objects referencing this SRS.

The method does the same thing as the C function **OSRReference()** (p. ??).



**Returns**

the updated reference count.

Referenced by OGRGeometry::assignSpatialReference(), and OGRProxiedLayer::GetSpatialRef().

**12.79.3.70 void OGRSpatialReference::Release ( )**

Decrements the reference count by one, and destroy if zero.

The method does the same thing as the C function **OSRRelease()** (p. ??).

References Dereference().

Referenced by OGRGeometry::assignSpatialReference().

**12.79.3.71 OGRErr OGRSpatialReference::SetACEA ( double *dfStdP1*, double *dfStdP2*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing* )**

Albers Conic Equal Area

References SetNormProjParm(), and SetProjection().

Referenced by importFromESRI(), importFromOzi(), importFromPCI(), importFromProj4(), and importFromUSGS().

**12.79.3.72 OGRErr OGRSpatialReference::SetAE ( double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing* )**

Azimuthal Equidistant

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

**12.79.3.73 OGRErr OGRSpatialReference::SetAngularUnits ( const char \* *pszUnitsName*, double *dfInRadians* )**

Set the angular units for the geographic coordinate system.

This method creates a UNIT subnode with the specified values as a child of the GEOGCS node.

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

**Parameters**

<i>pszUnitsName</i>	the units name to be used. Some preferred units names can be found in <b>ogr_srs_api.h</b> (p. ??) such as SRS_UA_DEGREE.
<i>dfInRadians</i>	the value to multiply by an angle in the indicated units to transform to radians. Some standard conversion factors can be found in <b>ogr_srs_api.h</b> (p. ??).

**Returns**

OGRERR\_NONE on success.

References OGR\_SRSNode::AddChild(), OGR\_SRSNode::FindChild(), GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), and OGR\_SRSNode::SetValue().

Referenced by CloneGeogCS(), Fixup(), and importFromPCI().

**12.79.3.74 OGRErr OGRSpatialReference::SetAuthority ( const char \* *pszTargetKey*, const char \* *pszAuthority*, int *nCode* )**

Set the authority for a node.

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

## Parameters

<i>pszTargetKey</i>	the partial or complete path to the node to set an authority on. ie. "PROJCS", "GEOGCS" or "GEOGCS UNIT".
<i>pszAuthority</i>	authority name, such as "EPSG".
<i>nCode</i>	code for value with this authority.

## Returns

OGRERR\_NONE on success.

References OGR\_SRSNode::AddChild(), OGR\_SRSNode::DestroyChild(), OGR\_SRSNode::FindChild(), and GetAttrNode().

Referenced by AutoIdentifyEPSG(), importFromEPSGA(), importFromPanorama(), importFromPCI(), importFromUSGS(), and importFromWMSAUTO().

**12.79.3.75** OGRErr OGRSpatialReference::SetAxes ( const char \* *pszTargetKey*, const char \* *pszXAxisName*, OGRAxisOrientation *eXAxisOrientation*, const char \* *pszYAxisName*, OGRAxisOrientation *eYAxisOrientation* )

Set the axes for a coordinate system.

Set the names, and orientations of the axes for either a projected (PROJCS) or geographic (GEOGCS) coordinate system.

This method is equivalent to the C function OSRSetAxes().

## Parameters

<i>pszTargetKey</i>	either "PROJCS" or "GEOGCS", must already exist in SRS.
<i>pszXAxisName</i>	name of first axis, normally "Long" or "Easting".
<i>eXAxisOrientation</i>	normally OAO_East.
<i>pszYAxisName</i>	name of second axis, normally "Lat" or "Northing".
<i>eYAxisOrientation</i>	normally OAO_North.

## Returns

OGRERR\_NONE on success or an error code.

References OGR\_SRSNode::AddChild(), OGR\_SRSNode::DestroyChild(), OGR\_SRSNode::FindChild(), and OSRAxisEnumToName().

**12.79.3.76** OGRErr OGRSpatialReference::SetBonne ( double *dfStdP1*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing* )

Bonne

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

**12.79.3.77** OGRErr OGRSpatialReference::SetCEA ( double *dfStdP1*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing* )

Cylindrical Equal Area

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), and importFromProj4().

**12.79.3.78** `OGRERR OGRSpatialReference::SetCompoundCS ( const char * pszName, const OGRSpatialReference * poHorizSRS, const OGRSpatialReference * poVertSRS )`

Setup a compound coordinate system.

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

This method is replace the current SRS with a COMPD\_CS coordinate system consisting of the passed in horizontal and vertical coordinate systems.

#### Parameters

<i>pszName</i>	the name of the compound coordinate system.
<i>poHorizSRS</i>	the horizontal SRS (PROJCS or GEOGCS).
<i>poVertSRS</i>	the vertical SRS (VERT_CS).

#### Returns

OGRERR\_NONE on success.

References `OGR_SRSNode::AddChild()`, `Clear()`, `OGR_SRSNode::Clone()`, `CPLError()`, `IsGeographic()`, `IsProjected()`, and `IsVertical()`.

**12.79.3.79** `OGRERR OGRSpatialReference::SetCS ( double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing )`

Cassini-Soldner

References `SetNormProjParm()`, and `SetProjection()`.

Referenced by `importFromPCI()`, and `importFromProj4()`.

**12.79.3.80** `OGRERR OGRSpatialReference::SetEC ( double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing )`

Equidistant Conic

References `SetNormProjParm()`, and `SetProjection()`.

Referenced by `importFromESRI()`, `importFromPanorama()`, `importFromPCI()`, `importFromProj4()`, and `importFromUSGS()`.

**12.79.3.81** `OGRERR OGRSpatialReference::SetEckert ( int nVariation, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing )`

Eckert I-VI

References `CPLError()`, `SetNormProjParm()`, and `SetProjection()`.

Referenced by `importFromProj4()`.

**12.79.3.82** `OGRERR OGRSpatialReference::SetEquirectangular ( double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing )`

Equirectangular

References `SetNormProjParm()`, and `SetProjection()`.

Referenced by `importFromPanorama()`, `importFromProj4()`, and `importFromWMSAUTO()`.

12.79.3.83 OGRErr OGRSpatialReference::SetEquirectangular2 ( double *dfCenterLat*, double *dfCenterLong*, double *dfPseudoStdParallel1*, double *dfFalseEasting*, double *dfFalseNorthing* )

Equirectangular generalized form :

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), importFromProj4(), and importFromUSGS().

12.79.3.84 OGRErr OGRSpatialReference::SetExtension ( const char \* *pszTargetKey*, const char \* *pszName*, const char \* *pszValue* )

Set extension value.

Set the value of the named EXTENSION item for the identified target node.

Parameters

<i>pszTargetKey</i>	the name or path to the parent node of the EXTENSION.
<i>pszName</i>	the name of the extension being fetched.
<i>pszValue</i>	the value to set

Returns

OGRERR\_NONE on success

References OGR\_SRSNode::AddChild(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), OGR\_SRSNode::GetValue(), and OGR\_SRSNode::SetValue().

Referenced by importFromProj4().

12.79.3.85 OGRErr OGRSpatialReference::SetFromUserInput ( const char \* *pszDefinition* )

Set spatial reference from various text formats.

This method will examine the provided input, and try to deduce the format, and then use it to initialize the spatial reference system. It may take the following forms:

1. Well Known Text definition - passed on to **importFromWkt()** (p. ??).
2. "EPSG:n" - number passed on to **importFromEPSG()** (p. ??).
3. "EPSGA:n" - number passed on to **importFromEPSGA()** (p. ??).
4. "AUTO:proj\_id,unit\_id,lon0,lat0" - WMS auto projections.
5. "urn:ogc:def:crs:EPSG::n" - ogc urns
6. PROJ.4 definitions - passed on to **importFromProj4()** (p. ??).
7. filename - file read for WKT, XML or PROJ.4 definition.
8. well known name accepted by **SetWellKnownGeogCS()** (p. ??), such as NAD27, NAD83, WGS84 or WGS72.
9. WKT (directly or in a file) in ESRI format should be prefixed with ESRI:: to trigger an automatic **morphFromESRI()** (p. ??).
10. "IGNF:xxx" - "+init=IGNF:xxx" passed on to **importFromProj4()** (p. ??).

It is expected that this method will be extended in the future to support XML and perhaps a simplified "minilanguage" for indicating common UTM and State Plane definitions.

This method is intended to be flexible, but by it's nature it is imprecise as it must guess information about the format intended. When possible applications should call the specific method appropriate if the input is known to be in a particular format.

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

## Parameters

<i>pszDefinition</i>	text definition to try to deduce SRS from.
----------------------	--

## Returns

OGRERR\_NONE on success, or an error code if the name isn't recognised, the definition is corrupt, or an EPSG value can't be successfully looked up.

References OGR\_SRSNode::AddChild(), Clear(), OGR\_SRSNode::Clone(), CPLDebug(), CPLMalloc(), CPLStrdup(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetValue(), importFromCRSURL(), importFromDict(), importFromEPSG(), importFromEPSGA(), importFromProj4(), importFromUrl(), importFromURN(), importFromWkt(), importFromWMSAUTO(), importFromXML(), morphFromESRI(), SetNode(), and SetWellKnownGeogCS().

Referenced by importFromUrl().

**12.79.3.86** OGRErr OGRSpatialReference::SetGaussSchreiberTMercator ( double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing* )

Gauss Schreiber Transverse Mercator

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

**12.79.3.87** OGRErr OGRSpatialReference::SetGeocCS ( const char \* *pszName* )

Set the user visible GEOCCS name.

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

This method will ensure a GEOCCS node is created as the root, and set the provided name on it. If used on a GEOGCS coordinate system, the DATUM and PRIMEM nodes from the GEOGCS will be transferred over to the GEOGCS.

## Parameters

<i>pszName</i>	the user visible name to assign. Not used as a key.
----------------	---

## Returns

OGRERR\_NONE on success.

## Since

OGR 1.9.0

References OGR\_SRSNode::Clone(), CPLDebug(), GetAttrNode(), OGR\_SRSNode::GetNode(), OGR\_SRSNode::GetValue(), OGR\_SRSNode::InsertChild(), and SetNode().

Referenced by importFromProj4().

**12.79.3.88** OGRErr OGRSpatialReference::SetGeogCS ( const char \* *pszGeogName*, const char \* *pszDatumName*, const char \* *pszSpheroidName*, double *dfSemiMajor*, double *dfInvFlattening*, const char \* *pszPMName* = NULL, double *dfPMOffset* = 0.0, const char \* *pszAngularUnits* = NULL, double *dfConvertToRadians* = 0.0 )

Set geographic coordinate system.

This method is used to set the datum, ellipsoid, prime meridian and angular units for a geographic coordinate system. It can be used on it's own to establish a geographic spatial reference, or applied to a projected coordinate system to establish the underlying geographic coordinate system.

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

## Parameters

<i>pszGeogName</i>	user visible name for the geographic coordinate system (not to serve as a key).
<i>pszDatumName</i>	key name for this datum. The OpenGIS specification lists some known values, and otherwise EPSG datum names with a standard transformation are considered legal keys.
<i>pszSpheroid-Name</i>	user visible spheroid name (not to serve as a key)
<i>dfSemiMajor</i>	the semi major axis of the spheroid.
<i>dfInvFlattening</i>	the inverse flattening for the spheroid. This can be computed from the semi minor axis as $1/f = 1.0 / (1.0 - \text{semiminor}/\text{semimajor})$ .
<i>pszPMName</i>	the name of the prime meridian (not to serve as a key) If this is NULL a default value of "Greenwich" will be used.
<i>dfPMOffset</i>	the longitude of greenwich relative to this prime meridian.
<i>pszAngularUnits</i>	the angular units name (see <b>ogr_srs_api.h</b> (p. ??) for some standard names). If NULL a value of "degrees" will be assumed.
<i>dfConvertTo-Radians</i>	value to multiply angular units by to transform them to radians. A value of SRS_UL_DEGREE_CONV will be used if pszAngularUnits is NULL.

## Returns

OGRERR\_NONE on success.

References OGR\_SRSNode::AddChild(), Clear(), CopyGeogCSFrom(), CPLAtof(), OGR\_SRSNode::DestroyChild(), OGR\_SRSNode::FindChild(), GetAttrNode(), OGR\_SRSNode::InsertChild(), IsGeocentric(), SetGeogCS(), and SetRoot().

Referenced by importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), importFromUSGS(), and SetGeogCS().

12.79.3.89 OGRErr OGRSpatialReference::SetGEOS ( double *dfCentralMeridian*, double *dfSatelliteHeight*, double *dfFalseEasting*, double *dfFalseNorthing* )

Geostationary Satellite

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

12.79.3.90 OGRErr OGRSpatialReference::SetGH ( double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing* )

Goode Homolosine

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

12.79.3.91 OGRErr OGRSpatialReference::SetGnomonic ( double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing* )

Gnomonic

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

12.79.3.92 OGRErr OGRSpatialReference::SetGS ( double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing* )

Gall Stereographic

References SetNormProjParm(), and SetProjection().



Referenced by `importFromProj4()`.

**12.79.3.93** `OGRERR OGRSpatialReference::SetHOM ( double dfCenterLat, double dfCenterLong, double dfAzimuth, double dfRectToSkew, double dfScale, double dfFalseEasting, double dfFalseNorthing )`

Set a Hotine Oblique Mercator projection using azimuth angle.

Hotine Oblique Mercator

This projection corresponds to EPSG projection method 9812, also sometimes known as hotine oblique mercator (variant A)..

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

#### Parameters

<i>dfCenterLat</i>	Latitude of the projection origin.
<i>dfCenterLong</i>	Longitude of the projection origin.
<i>dfAzimuth</i>	Azimuth, measured clockwise from North, of the projection centerline.
<i>dfRectToSkew</i>	?
<i>dfScale</i>	Scale factor applies to the projection origin.
<i>dfFalseEasting</i>	False easting.
<i>dfFalseNorthing</i>	False northing.

#### Returns

OGRERR\_NONE on success.

References `SetNormProjParm()`, and `SetProjection()`.

Referenced by `importFromPCI()`, `importFromProj4()`, and `importFromUSGS()`.

**12.79.3.94** `OGRERR OGRSpatialReference::SetHOM2PNO ( double dfCenterLat, double dfLat1, double dfLong1, double dfLat2, double dfLong2, double dfScale, double dfFalseEasting, double dfFalseNorthing )`

Set a Hotine Oblique Mercator projection using two points on projection centerline.

This method does the same thing as the C function **OSRSetHOM2PNO()** (p. ??).

#### Parameters

<i>dfCenterLat</i>	Latitude of the projection origin.
<i>dfLat1</i>	Latitude of the first point on center line.
<i>dfLong1</i>	Longitude of the first point on center line.
<i>dfLat2</i>	Latitude of the second point on center line.
<i>dfLong2</i>	Longitude of the second point on center line.
<i>dfScale</i>	Scale factor applies to the projection origin.
<i>dfFalseEasting</i>	False easting.
<i>dfFalseNorthing</i>	False northing.

#### Returns

OGRERR\_NONE on success.

References `SetNormProjParm()`, and `SetProjection()`.

Referenced by `importFromPCI()`, and `importFromUSGS()`.

**12.79.3.95** `OGRERR OGRSpatialReference::SetHOMAC ( double dfCenterLat, double dfCenterLong, double dfAzimuth, double dfRectToSkew, double dfScale, double dfFalseEasting, double dfFalseNorthing )`

Set an Hotine Oblique Mercator Azimuth Center projection using azimuth angle.

### Hotine Oblique Mercator Azimuth Center / Variant B

This projection corresponds to EPSG projection method 9815, also sometimes known as hotine oblique mercator (variant B).

This method does the same thing as the C function `OSRSetHOMAC()`.

#### Parameters

<i>dfCenterLat</i>	Latitude of the projection origin.
<i>dfCenterLong</i>	Longitude of the projection origin.
<i>dfAzimuth</i>	Azimuth, measured clockwise from North, of the projection centerline.
<i>dfRectToSkew</i>	?
<i>dfScale</i>	Scale factor applies to the projection origin.
<i>dfFalseEasting</i>	False easting.
<i>dfFalseNorthing</i>	False northing.

#### Returns

`OGRERR_NONE` on success.

References `SetNormProjParm()`, and `SetProjection()`.

Referenced by `importFromProj4()`.

### 12.79.3.96 OGRErr OGRSpatialReference::SetIGH ( )

Interrupted Goode Homolosine

References `SetProjection()`.

Referenced by `importFromProj4()`.

### 12.79.3.97 OGRErr OGRSpatialReference::SetIWMPolyconic ( double *dfLat1*, double *dfLat2*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing* )

International Map of the World Polyconic

References `SetNormProjParm()`, and `SetProjection()`.

Referenced by `importFromPanorama()`, and `importFromProj4()`.

### 12.79.3.98 OGRErr OGRSpatialReference::SetKrovak ( double *dfCenterLat*, double *dfCenterLong*, double *dfAzimuth*, double *dfPseudoStdParallelLat*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing* )

Krovak Oblique Conic Conformal

References `SetNormProjParm()`, and `SetProjection()`.

Referenced by `importFromProj4()`.

### 12.79.3.99 OGRErr OGRSpatialReference::SetLAEA ( double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing* )

Lambert Azimuthal Equal-Area

References `SetNormProjParm()`, and `SetProjection()`.

Referenced by `importFromESRI()`, `importFromPanorama()`, `importFromPCI()`, `importFromProj4()`, and `importFrom-USGS()`.

**12.79.3.100** OGRErr OGRSpatialReference::SetLCC ( double *dfStdP1*, double *dfStdP2*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing* )

Lambert Conformal Conic

References SetNormProjParm(), and SetProjection().

Referenced by importFromESRI(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

**12.79.3.101** OGRErr OGRSpatialReference::SetLCC1SP ( double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing* )

Lambert Conformal Conic 1SP

References SetNormProjParm(), and SetProjection().

Referenced by importFromOzi(), importFromPCI(), and importFromProj4().

**12.79.3.102** OGRErr OGRSpatialReference::SetLCCB ( double *dfStdP1*, double *dfStdP2*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing* )

Lambert Conformal Conic (Belgium)

References SetNormProjParm(), and SetProjection().

**12.79.3.103** OGRErr OGRSpatialReference::SetLinearUnits ( const char \* *pszUnitsName*, double *dfInMeters* )

Set the linear units for the projection.

This method creates a UNIT subnode with the specified values as a child of the PROJCS, GEOCCS or LOCAL\_CS node.

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

#### Parameters

<i>pszUnitsName</i>	the units name to be used. Some preferred units names can be found in <b>ogr_srs_api.h</b> (p. ??) such as SRS_UL_METER, SRS_UL_FOOT and SRS_UL_US_FOOT.
<i>dfInMeters</i>	the value to multiple by a length in the indicated units to transform to meters. Some standard conversion factors can be found in <b>ogr_srs_api.h</b> (p. ??).

#### Returns

OGRERR\_NONE on success.

References SetTargetLinearUnits().

Referenced by Fixup(), importFromERM(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), importFromUSGS(), importFromWMSAUTO(), SetLinearUnitsAndUpdateParameters(), SetStatePlane(), and SetUTM().

**12.79.3.104** OGRErr OGRSpatialReference::SetLinearUnitsAndUpdateParameters ( const char \* *pszName*, double *dfInMeters* )

Set the linear units for the projection.

This method creates a UNIT subnode with the specified values as a child of the PROJCS or LOCAL\_CS node. It works the same as the **SetLinearUnits()** (p. ??) method, but it also updates all existing linear projection parameter values from the old units to the new units.

## Parameters

<i>pszName</i>	the units name to be used. Some preferred units names can be found in <b>ogr_srs_api.h</b> (p. ??) such as SRS_UL_METER, SRS_UL_FOOT and SRS_UL_US_FOOT.
<i>dfInMeters</i>	the value to multiple by a length in the indicated units to transform to meters. Some standard conversion factors can be found in <b>ogr_srs_api.h</b> (p. ??).

## Returns

OGRERR\_NONE on success.

References CPLStrdup(), GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), GetLinearUnits(), GetProjParm(), OGR\_SRSNode::GetValue(), SetLinearUnits(), and SetProjParm().

Referenced by importFromESRI(), and importFromPCI().

#### 12.79.3.105 OGRErr OGRSpatialReference::SetLocalCS ( const char \* *pszName* )

Set the user visible LOCAL\_CS name.

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

This method will ensure a LOCAL\_CS node is created as the root, and set the provided name on it. It must be used before **SetLinearUnits()** (p. ??).

## Parameters

<i>pszName</i>	the user visible name to assign. Not used as a key.
----------------	---

## Returns

OGRERR\_NONE on success.

References CPLDebug(), GetAttrNode(), and SetNode().

Referenced by importFromESRI(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromUSGS(), and SetStatePlane().

#### 12.79.3.106 OGRErr OGRSpatialReference::SetMC ( double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing* )

Miller Cylindrical

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), importFromProj4(), and importFromUSGS().

#### 12.79.3.107 OGRErr OGRSpatialReference::SetMercator ( double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing* )

Mercator

References SetNormProjParm(), and SetProjection().

Referenced by importFromESRI(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

#### 12.79.3.108 OGRErr OGRSpatialReference::SetMollweide ( double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing* )

Mollweide

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromProj4(), importFromUSGS(), and importFromWMSAUTO().

### 12.79.3.109 OGRErr OGRSpatialReference::SetNode ( const char \* *pszNodePath*, const char \* *pszNewNodeValue* )

Set attribute value in spatial reference.

Missing intermediate nodes in the path will be created if not already in existence. If the attribute has no children one will be created and assigned the value otherwise the zeroth child will be assigned the value.

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

#### Parameters

<i>pszNodePath</i>	full path to attribute to be set. For instance "PROJCS GEOGCS UNIT".
<i>pszNewNodeValue</i>	value to be assigned to node, such as "meter". This may be NULL if you just want to force creation of the intermediate path.

#### Returns

OGRERR\_NONE on success.

References OGR\_SRSNode::AddChild(), CSLCount(), CSLDestroy(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), OGR\_SRSNode::GetValue(), SetRoot(), and OGR\_SRSNode::SetValue().

Referenced by importFromCRSURL(), importFromProj4(), importFromURN(), morphFromESRI(), morphToESRI(), SetFromUserInput(), SetGeocCS(), SetLocalCS(), SetProjCS(), SetProjection(), and SetUTM().

### 12.79.3.110 OGRErr OGRSpatialReference::SetNormProjParm ( const char \* *pszName*, double *dfValue* )

Set a projection parameter with a normalized value.

This method is the same as **SetProjParm()** (p. ??) except that the value of the parameter passed in is assumed to be in "normalized" form (decimal degrees for angular values, meters for linear values. The values are converted in a form suitable for the GEOGCS and linear units in effect.

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

#### Parameters

<i>pszName</i>	the parameter name, which should be selected from the macros in <b>ogr_srs_api.h</b> (p. ??), such as SRS_PP_CENTRAL_MERIDIAN.
<i>dfValue</i>	value to assign.

#### Returns

OGRERR\_NONE on success.

References SetProjParm().

Referenced by importFromProj4(), SetACEA(), SetAE(), SetBonne(), SetCEA(), SetCS(), SetEC(), SetEckert(), SetEquirectangular(), SetEquirectangular2(), SetGaussSchreiberTMercator(), SetGEOS(), SetGH(), SetGnomonic(), SetGS(), SetHOM(), SetHOM2PNO(), SetHOMAC(), SetIWMPolyconic(), SetKrovak(), SetLAEA(), SetLCC(), SetLCC1SP(), SetLCCB(), SetMC(), SetMercator(), SetMollweide(), SetNZMG(), SetOrthographic(), SetOS(), SetPolyconic(), SetPS(), SetRobinson(), SetSinusoidal(), SetSOC(), SetStatePlane(), SetStereographic(), SetTM(), SetTMG(), SetTMSO(), SetTMVariant(), SetTPED(), SetUTM(), SetVDG(), and SetWagner().

### 12.79.3.111 OGRErr OGRSpatialReference::SetNZMG ( double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing* )

New Zealand Map Grid

References `SetNormProjParm()`, and `SetProjection()`.

Referenced by `importFromProj4()`.

**12.79.3.112** `OGRERR OGRSpatialReference::SetOrthographic ( double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing )`

Orthographic

References `SetNormProjParm()`, and `SetProjection()`.

Referenced by `importFromPCI()`, `importFromProj4()`, `importFromUSGS()`, and `importFromWMSAUTO()`.

**12.79.3.113** `OGRERR OGRSpatialReference::SetOS ( double dfOriginLat, double dfCMeridian, double dfScale, double dfFalseEasting, double dfFalseNorthing )`

Oblique Stereographic

References `SetNormProjParm()`, and `SetProjection()`.

Referenced by `importFromPCI()`, and `importFromProj4()`.

**12.79.3.114** `OGRERR OGRSpatialReference::SetPolyconic ( double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing )`

Polyconic

References `SetNormProjParm()`, and `SetProjection()`.

Referenced by `importFromESRI()`, `importFromPanorama()`, `importFromPCI()`, `importFromProj4()`, and `importFromUSGS()`.

**12.79.3.115** `OGRERR OGRSpatialReference::SetProjCS ( const char * pszName )`

Set the user visible PROJCS name.

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

This method will ensure a PROJCS node is created as the root, and set the provided name on it. If used on a GEOGCS coordinate system, the GEOGCS node will be demoted to be a child of the new PROJCS root.

Parameters

<i>pszName</i>	the user visible name to assign. Not used as a key.
----------------	---

Returns

`OGRERR_NONE` on success.

References `CPLDebug()`, `GetAttrNode()`, `OGR_SRSNode::GetValue()`, `OGR_SRSNode::InsertChild()`, and `SetNode()`.

**12.79.3.116** `OGRERR OGRSpatialReference::SetProjection ( const char * pszProjection )`

Set a projection name.

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

## Parameters

<i>pszProjection</i>	the projection name, which should be selected from the macros in <b>ogr_srs_api.h</b> (p. ??), such as SRS_PT_TRANSVERSE_MERCATOR.
----------------------	--

## Returns

OGRERR\_NONE on success.

References GetAttrNode(), OGR\_SRSNode::GetValue(), OGR\_SRSNode::InsertChild(), and SetNode().

Referenced by importFromProj4(), SetACEA(), SetAE(), SetBonne(), SetCEA(), SetCS(), SetEC(), SetEckert(), SetEquirectangular(), SetEquirectangular2(), SetGaussSchreiberTMercator(), SetGEOS(), SetGH(), SetGnomonic(), SetGS(), SetHOM(), SetHOM2PNO(), SetHOMAC(), SetIGH(), SetIWMPolyconic(), SetKrovak(), SetLAEA(), SetLCC(), SetLCC1SP(), SetLCCB(), SetMC(), SetMercator(), SetMollweide(), SetNZMG(), SetOrthographic(), SetOS(), SetPolyconic(), SetPS(), SetRobinson(), SetSinusoidal(), SetSOC(), SetStereographic(), SetTM(), SetTMG(), SetTMSO(), SetTMVariant(), SetTPED(), SetUTM(), SetVDG(), and SetWagner().

### 12.79.3.117 OGRErr OGRSpatialReference::SetProjParm ( const char \* *pszParmName*, double *dfValue* )

Set a projection parameter value.

Adds a new PARAMETER under the PROJCS with the indicated name and value.

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

Please check [http://www.remotesensing.org/geotiff/proj\\_list](http://www.remotesensing.org/geotiff/proj_list) pages for legal parameter names for specific projections.

## Parameters

<i>pszParmName</i>	the parameter name, which should be selected from the macros in <b>ogr_srs_api.h</b> (p. ??), such as SRS_PP_CENTRAL_MERIDIAN.
<i>dfValue</i>	value to assign.

## Returns

OGRERR\_NONE on success.

References OGR\_SRSNode::AddChild(), GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), OGR\_SRSNode::GetValue(), and OGR\_SRSNode::SetValue().

Referenced by morphFromESRI(), SetLinearUnitsAndUpdateParameters(), and SetNormProjParm().

### 12.79.3.118 OGRErr OGRSpatialReference::SetPS ( double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing* )

Polar Stereographic

References SetNormProjParm(), and SetProjection().

Referenced by importFromESRI(), importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

### 12.79.3.119 OGRErr OGRSpatialReference::SetRobinson ( double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing* )

Robinson

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), importFromProj4(), and importFromUSGS().

12.79.3.120 void OGRSpatialReference::SetRoot ( OGR\_SRSNode \* *poNewRoot* )

Set the root SRS node.

If the object has an existing tree of OGR\_SRSNodes, they are destroyed as part of assigning the new root. Ownership of the passed **OGR\_SRSNode** (p. ??) is assumed by the **OGRSpatialReference** (p. ??).

#### Parameters

<i>poNewRoot</i>	object to assign as root.
------------------	---------------------------

Referenced by CloneGeogCS(), CopyGeogCSFrom(), SetGeogCS(), SetNode(), SetVertCS(), and StripVertical().

12.79.3.121 OGRErr OGRSpatialReference::SetSinusoidal ( double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing* )

Sinusoidal

References SetNormProjParm(), and SetProjection().

Referenced by importFromOzi(), importFromPCI(), importFromProj4(), and importFromUSGS().

12.79.3.122 OGRErr OGRSpatialReference::SetSOC ( double *dfLatitudeOfOrigin*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing* )

Swiss Oblique Cylindrical

References SetNormProjParm(), and SetProjection().

12.79.3.123 OGRErr OGRSpatialReference::SetStatePlane ( int *nZone*, int *bNAD83* = TRUE, const char \* *pszOverrideUnitName* = NULL, double *dfOverrideUnit* = 0.0 )

Set State Plane projection definition.

State Plane

This will attempt to generate a complete definition of a state plane zone based on generating the entire SRS from the EPSG tables. If the EPSG tables are unavailable, it will produce a stubbed LOCAL\_CS definition and return OGRERR\_FAILURE.

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

#### Parameters

<i>nZone</i>	State plane zone number, in the USGS numbering scheme (as distinct from the Arc/Info and Erdas numbering scheme).
<i>bNAD83</i>	TRUE if the NAD83 zone definition should be used or FALSE if the NAD27 zone definition should be used.
<i>pszOverrideUnitName</i>	Linear unit name to apply overriding the legal definition for this zone.
<i>dfOverrideUnit</i>	Linear unit conversion factor to apply overriding the legal definition for this zone.

#### Returns

OGRERR\_NONE on success, or OGRERR\_FAILURE on failure, mostly likely due to the EPSG tables not being accessible.

References Clear(), CPLAtof(), CPLError(), OGR\_SRSNode::DestroyChild(), OGR\_SRSNode::FindChild(), GetAttrNode(), GetLinearUnits(), GetNormProjParm(), importFromEPSG(), SetLinearUnits(), SetLocalCS(), and SetNormProjParm().

Referenced by importFromESRI(), importFromPCI(), and importFromUSGS().



12.79.3.124 OGRErr OGRSpatialReference::SetStereographic ( double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing* )

Stereographic

References SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromPCI(), importFromProj4(), and importFromUSGS().

12.79.3.125 OGRErr OGRSpatialReference::SetTargetLinearUnits ( const char \* *pszTargetKey*, const char \* *pszUnitsName*, double *dfInMeters* )

Set the linear units for the projection.

This method creates a UNIT subnode with the specified values as a child of the target node.

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

Parameters

<i>pszTargetKey</i>	the keyword to set the linear units for. ie. "PROJCS" or "VERT_CS"
<i>pszUnitsName</i>	the units name to be used. Some preferred units names can be found in <b>ogr_srs_api.h</b> (p. ??) such as SRS_UL_METER, SRS_UL_FOOT and SRS_UL_US_FOOT.
<i>dfInMeters</i>	the value to multiple by a length in the indicated units to transform to meters. Some standard conversion factors can be found in <b>ogr_srs_api.h</b> (p. ??).

Returns

OGRERR\_NONE on success.

Since

OGR 1.9.0

References OGR\_SRSNode::AddChild(), OGR\_SRSNode::DestroyChild(), OGR\_SRSNode::FindChild(), GetAttrNode(), OGR\_SRSNode::GetChild(), OGR\_SRSNode::GetChildCount(), IsVertical(), and OGR\_SRSNode::SetValue().

Referenced by SetLinearUnits().

12.79.3.126 OGRErr OGRSpatialReference::SetTM ( double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing* )

Transverse Mercator

References SetNormProjParm(), and SetProjection().

Referenced by importFromESRI(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), importFromUSGS(), and importFromWMSAUTO().

12.79.3.127 OGRErr OGRSpatialReference::SetTMG ( double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing* )

Tunesia Mining Grid

References SetNormProjParm(), and SetProjection().

12.79.3.128 OGRErr OGRSpatialReference::SetTMSO ( double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing* )

Transverse Mercator (South Oriented)

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

**12.79.3.129** OGRErr OGRSpatialReference::SetTMVariant ( const char \* *pszVariantName*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing* )

Transverse Mercator variants.

References SetNormProjParm(), and SetProjection().

**12.79.3.130** OGRErr OGRSpatialReference::SetTOWGS84 ( double *dfDX*, double *dfDY*, double *dfDZ*, double *dfEX* = 0.0, double *dfEY* = 0.0, double *dfEZ* = 0.0, double *dfPPM* = 0.0 )

Set the Bursa-Wolf conversion to WGS84.

This will create the TOWGS84 node as a child of the DATUM. It will fail if there is no existing DATUM node. Unlike most **OGRSpatialReference** (p. ??) methods it will insert itself in the appropriate order, and will replace an existing TOWGS84 node if there is one.

The parameters have the same meaning as EPSG transformation 9606 (Position Vector 7-param. transformation).

This method is the same as the C function **OSRSetTOWGS84()** (p. ??).

#### Parameters

<i>dfDX</i>	X child in meters.
<i>dfDY</i>	Y child in meters.
<i>dfDZ</i>	Z child in meters.
<i>dfEX</i>	X rotation in arc seconds (optional, defaults to zero).
<i>dfEY</i>	Y rotation in arc seconds (optional, defaults to zero).
<i>dfEZ</i>	Z rotation in arc seconds (optional, defaults to zero).
<i>dfPPM</i>	scaling factor (parts per million).

#### Returns

OGRErr\_NONE on success.

References OGR\_SRSNode::AddChild(), OGR\_SRSNode::DestroyChild(), OGR\_SRSNode::FindChild(), GetAttrNode(), OGR\_SRSNode::GetChildCount(), and OGR\_SRSNode::InsertChild().

Referenced by importFromOzi(), importFromPCI(), and importFromProj4().

**12.79.3.131** OGRErr OGRSpatialReference::SetTPED ( double *dfLat1*, double *dfLong1*, double *dfLat2*, double *dfLong2*, double *dfFalseEasting*, double *dfFalseNorthing* )

Two Point Equidistant

References SetNormProjParm(), and SetProjection().

Referenced by importFromProj4().

**12.79.3.132** OGRErr OGRSpatialReference::SetUTM ( int *nZone*, int *bNorth* = TRUE )

Set UTM projection definition.

Universal Transverse Mercator

This will generate a projection definition with the full set of transverse mercator projection parameters for the given UTM zone. If no PROJCS[] description is set yet, one will be set to look like "UTM Zone %d, {Northern, Southern} Hemisphere".

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

## Parameters

<i>nZone</i>	UTM zone.
<i>bNorth</i>	TRUE for northern hemisphere, or FALSE for southern hemisphere.

## Returns

OGRERR\_NONE on success.

References GetAttrValue(), SetLinearUnits(), SetNode(), SetNormProjParm(), and SetProjection().

Referenced by importFromESRI(), importFromOzi(), importFromPanorama(), importFromPCI(), importFromProj4(), importFromUSGS(), and importFromWMSAUTO().

**12.79.3.133 OGRErr OGRSpatialReference::SetVDG ( double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing* )**

VanDerGrinten

References SetNormProjParm(), and SetProjection().

Referenced by importFromPCI(), importFromProj4(), and importFromUSGS().

**12.79.3.134 OGRErr OGRSpatialReference::SetVertCS ( const char \* *pszVertCSName*, const char \* *pszVertDatumName*, int *nVertDatumType* = 2005 )**

Set the user visible VERT\_CS name.

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

This method will ensure a VERT\_CS node is created if needed. If the existing coordinate system is GEOGCS or PROJCS rooted, then it will be turned into a COMPD\_CS.

## Parameters

<i>pszVertCSName</i>	the user visible name of the vertical coordinate system. Not used as a key.
<i>pszVertDatumName</i>	the user visible name of the vertical datum. It is helpful if this matches the EPSG name.
<i>nVertDatumType</i>	the OGC vertical datum type, usually 2005.

## Returns

OGRERR\_NONE on success.

## Since

OGR 1.9.0

References OGR\_SRSNode::AddChild(), Clear(), GetAttrNode(), OGR\_SRSNode::GetValue(), IsGeographic(), IsProjected(), and SetRoot().

**12.79.3.135 OGRErr OGRSpatialReference::SetWagner ( int *nVariation*, double *dfCenterLat*, double *dfFalseEasting*, double *dfFalseNorthing* )**

Wagner I – VII

References CPLError(), SetNormProjParm(), and SetProjection().

Referenced by importFromPanorama(), importFromProj4(), and importFromUSGS().

12.79.3.136 OGRErr OGRSpatialReference::SetWellKnownGeogCS ( const char \* *pszName* )

Set a GeogCS based on well known name.

This may be called on an empty **OGRSpatialReference** (p. ??) to make a geographic coordinate system, or on something with an existing PROJCS node to set the underlying geographic coordinate system of a projected coordinate system.

The following well known text values are currently supported:

- "WGS84": same as "EPSG:4326" but has no dependence on EPSG data files.
- "WGS72": same as "EPSG:4322" but has no dependence on EPSG data files.
- "NAD27": same as "EPSG:4267" but has no dependence on EPSG data files.
- "NAD83": same as "EPSG:4269" but has no dependence on EPSG data files.
- "EPSG:n": same as doing an ImportFromEPSG(n).

## Parameters

<i>pszName</i>	name of well known geographic coordinate system.
----------------	--

## Returns

OGRErr\_NONE on success, or OGRErr\_FAILURE if the name isn't recognised, the target object is already initialized, or an EPSG value can't be successfully looked up.

References CopyGeogCSFrom(), importFromEPSG(), importFromEPSGA(), importFromWkt(), and IsGeographic().

Referenced by importFromESRI(), importFromPanorama(), importFromProj4(), importFromUSGS(), importFromWMSAUTO(), and SetFromUserInput().

12.79.3.137 OGRErr OGRSpatialReference::StripCTParms ( OGR\_SRSNode \* *poCurrent* = NULL )

Strip OGC CT Parameters.

This method will remove all components of the coordinate system that are specific to the OGC CT Specification. That is it will attempt to strip it down to being compatible with the Simple Features 1.0 specification.

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

## Parameters

<i>poCurrent</i>	node to operate on. NULL to operate on whole tree.
------------------	--

## Returns

OGRErr\_NONE on success or an error code.

References OGR\_SRSNode::GetValue(), OGR\_SRSNode::StripNodes(), and StripVertical().

Referenced by morphFromESRI(), and morphToESRI().

## 12.79.3.138 OGRErr OGRSpatialReference::StripVertical ( )

Convert a compound cs into a horizontal CS.

If this SRS is of type COMPD\_CS[] then the vertical CS and the root COMPD\_CS nodes are stripped resulting and only the horizontal coordinate system portion remains (normally PROJCS, GEOGCS or LOCAL\_CS).

If this is not a compound coordinate system then nothing is changed.

Since

OGR 1.8.0

References OGR\_SRSNode::Clone(), OGR\_SRSNode::GetChild(), and SetRoot().

Referenced by StripCTParms().

#### 12.79.3.139 OGRErr OGRSpatialReference::Validate ( )

Validate SRS tokens.

This method attempts to verify that the spatial reference system is well formed, and consists of known tokens. The validation is not comprehensive.

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

Returns

OGRErr\_NONE if all is fine, OGRErr\_CORRUPT\_DATA if the SRS is not well formed, and OGRErr\_UN-  
SUPPORTED\_SRS if the SRS is well formed, but contains non-standard PROJECTION[] values.

References CPLDebug().

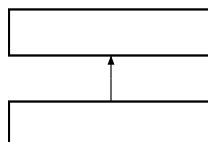
The documentation for this class was generated from the following files:

- **ogr\_spatialref.h**
- ogr\_fromepsg.cpp
- ogr\_srs\_dict.cpp
- ogr\_srs\_erm.cpp
- ogr\_srs\_esri.cpp
- ogr\_srs\_ozl.cpp
- ogr\_srs\_panorama.cpp
- ogr\_srs\_pci.cpp
- ogr\_srs\_proj4.cpp
- ogr\_srs\_usgs.cpp
- ogr\_srs\_validate.cpp
- ogr\_srs\_xml.cpp
- ogrspatialreference.cpp

## 12.80 OGRStyleBrush Class Reference

```
#include <ogr_featurestyle.h>
```

Inheritance diagram for OGRStyleBrush:



### 12.80.1 Detailed Description

This class represents a style brush

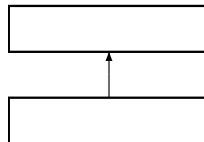
The documentation for this class was generated from the following files:

- **ogr\_featurestyle.h**
- ogrfeaturestyle.cpp

## 12.81 OGRStyleLabel Class Reference

```
#include <ogr_featurestyle.h>
```

Inheritance diagram for OGRStyleLabel:



### 12.81.1 Detailed Description

This class represents a style label

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

- **ogr\_featurestyle.h**
- ogrfeaturestyle.cpp

## 12.82 OGRStyleMgr Class Reference

```
#include <ogr_featurestyle.h>
```

### Public Member Functions

- **OGRStyleMgr** (**OGRStyleTable** \*poDataSetStyleTable=NULL)  
*Constructor.*
- **~OGRStyleMgr** ()  
*Destructor.*
- GBool **SetFeatureStyleString** (**OGRFeature** \*, const char \*pszStyleString=NULL, GBool bNoMatching=FALSE)  
*Set a style in a feature.*
- const char \* **InitFromFeature** (**OGRFeature** \*)  
*Initialize style manager from the style string of a feature.*
- GBool **InitStyleString** (const char \*pszStyleString=NULL)  
*Initialize style manager from the style string.*
- const char \* **GetStyleName** (const char \*pszStyleString=NULL)  
*Get the name of a style from the style table.*
- const char \* **GetStyleByName** (const char \*pszStyleName)  
*find a style in the current style table.*
- GBool **AddStyle** (const char \*pszStyleName, const char \*pszStyleString=NULL)  
*Add a style to the current style table.*
- const char \* **GetStyleString** (**OGRFeature** \*pFeature=NULL)  
*Get the style string from the style manager.*
- GBool **AddPart** (**OGRStyleTool** \*)

- Add a part (style tool) to the current style.*
- GBool **AddPart** (const char \*)  
*Add a part (style string) to the current style.*
- int **GetPartCount** (const char \*pszStyleString=NULL)  
*Get the number of parts in a style.*
- OGRStyleTool \* **GetPart** (int hPartId, const char \*pszStyleString=NULL)  
*Fetch a part (style tool) from the current style.*

### 12.82.1 Detailed Description

This class represents a style manager

### 12.82.2 Constructor & Destructor Documentation

#### 12.82.2.1 OGRStyleMgr::OGRStyleMgr ( OGRStyleTable \* poDataSetStyleTable = NULL )

Constructor.

This method is the same as the C function **OGR\_SM\_Create()** (p. ??)

Parameters

<i>poDataSetStyle- Table</i>	(currently unused, reserved for future use), pointer to <b>OGRStyleTable</b> (p. ??). Pass NULL for now.
----------------------------------	--

#### 12.82.2.2 OGRStyleMgr::~~OGRStyleMgr ( )

Destructor.

This method is the same as the C function **OGR\_SM\_Destroy()** (p. ??)

### 12.82.3 Member Function Documentation

#### 12.82.3.1 GBool OGRStyleMgr::AddPart ( OGRStyleTool \* poStyleTool )

Add a part (style tool) to the current style.

This method is the same as the C function **OGR\_SM\_AddPart()** (p. ??).

Parameters

<i>poStyleTool</i>	the style tool defining the part to add.
--------------------	--

Returns

TRUE on success, FALSE on errors.

References CPLStrdup().

#### 12.82.3.2 GBool OGRStyleMgr::AddPart ( const char \* pszPart )

Add a part (style string) to the current style.



## Parameters

<i>pszPart</i>	the style string defining the part to add.
----------------	--

## Returns

TRUE on success, FALSE on errors.

References CPLStrdup().

### 12.82.3.3 GBool OGRStyleMgr::AddStyle ( const char \* *pszStyleName*, const char \* *pszStyleString* = NULL )

Add a style to the current style table.

This method is the same as the C function **OGR\_SM\_AddStyle()** (p. ??).

## Parameters

<i>pszStyleName</i>	the name of the style to add.
<i>pszStyleString</i>	the style string to use, or NULL to use the style stored in the manager.

## Returns

TRUE on success, FALSE on errors.

References OGRStyleTable::AddStyle().

### 12.82.3.4 OGRStyleTool \* OGRStyleMgr::GetPart ( int *nPartId*, const char \* *pszStyleString* = NULL )

Fetch a part (style tool) from the current style.

This method is the same as the C function **OGR\_SM\_GetPart()** (p. ??).

This method instantiates a new object that should be freed with **OGR\_ST\_Destroy()** (p. ??).

## Parameters

<i>nPartId</i>	the part number (0-based index).
<i>pszStyleString</i>	(optional) the style string on which to operate. If NULL then the current style string stored in the style manager is used.

## Returns

**OGRStyleTool** (p. ??) of the requested part (style tools) or NULL on error.

References CSLDestroy(), and CSLTokenizeString2().

### 12.82.3.5 int OGRStyleMgr::GetPartCount ( const char \* *pszStyleString* = NULL )

Get the number of parts in a style.

This method is the same as the C function **OGR\_SM\_GetPartCount()** (p. ??).

## Parameters

<i>pszStyleString</i>	(optional) the style string on which to operate. If NULL then the current style string stored in the style manager is used.
-----------------------	---

**Returns**

the number of parts (style tools) in the style.

**12.82.3.6** `const char * OGRStyleMgr::GetStyleByName ( const char * pszStyleName )`

find a style in the current style table.

**Parameters**

<i>pszStyleName</i>	the name of the style to add.
---------------------	-------------------------------

**Returns**

the style string matching the name or NULL if not found or error.

References OGRStyleTable::Find().

Referenced by InitStyleString().

**12.82.3.7** `const char * OGRStyleMgr::GetStyleName ( const char * pszStyleString = NULL )`

Get the name of a style from the style table.

**Parameters**

<i>pszStyleString</i>	the style to search for, or NULL to use the style currently stored in the manager.
-----------------------	--

**Returns**

The name if found, or NULL on error.

References OGRStyleTable::GetStyleName().

Referenced by SetFeatureStyleString().

**12.82.3.8** `const char * OGRStyleMgr::GetStyleString ( OGRFeature * poFeature = NULL )`

Get the style string from the style manager.

**Parameters**

<i>poFeature</i>	feature object from which to read the style or NULL to get the style string stored in the manager.
------------------	--

**Returns**

the style string stored in the feature or the style string stored in the style manager if *poFeature* is NULL

NOTE: this method will call **OGRStyleMgr::InitFromFeature()** (p. ??) if *poFeature* is not NULL and replace the style string stored in the style manager

References InitFromFeature().

**12.82.3.9** `const char * OGRStyleMgr::InitFromFeature ( OGRFeature * poFeature )`

Initialize style manager from the style string of a feature.

This method is the same as the C function **OGR\_SM\_InitFromFeature()** (p. ??).

## Parameters

<i>poFeature</i>	feature object from which to read the style.
------------------	--

## Returns

a reference to the style string read from the feature, or NULL in case of error..

References OGRFeature::GetStyleString(), and InitStyleString().

Referenced by GetStyleString().

### 12.82.3.10 GBool OGRStyleMgr::InitStyleString ( const char \* *pszStyleString* = NULL )

Initialize style manager from the style string.

This method is the same as the C function **OGR\_SM\_InitStyleString()** (p. ??).

## Parameters

<i>pszStyleString</i>	the style string to use (can be NULL).
-----------------------	--

## Returns

TRUE on success, FALSE on errors.

References CPLStrdup(), and GetStyleByName().

Referenced by InitFromFeature().

### 12.82.3.11 GBool OGRStyleMgr::SetFeatureStyleString ( OGRFeature \* *poFeature*, const char \* *pszStyleString* = NULL, GBool *bNoMatching* = FALSE )

Set a style in a feature.

## Parameters

<i>poFeature</i>	the feature object to store the style in
<i>pszStyleString</i>	the style to store
<i>bNoMatching</i>	TRUE to lookup the style in the style table and add the name to the feature

## Returns

TRUE on success, FALSE on error.

References GetStyleName(), and OGRFeature::SetStyleString().

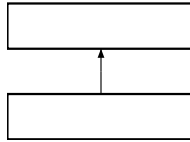
The documentation for this class was generated from the following files:

- **ogr\_featurestyle.h**
- ogrfeaturestyle.cpp

## 12.83 OGRStylePen Class Reference

```
#include <ogr_featurestyle.h>
```

Inheritance diagram for OGRStylePen:



### 12.83.1 Detailed Description

This class represents a style pen

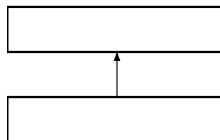
The documentation for this class was generated from the following files:

- **ogr\_featurestyle.h**
- ogrfeaturestyle.cpp

## 12.84 OGRStyleSymbol Class Reference

```
#include <ogr_featurestyle.h>
```

Inheritance diagram for OGRStyleSymbol:



### 12.84.1 Detailed Description

This class represents a style symbol

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

- **ogr\_featurestyle.h**
- ogrfeaturestyle.cpp

## 12.85 OGRStyleTable Class Reference

```
#include <ogr_featurestyle.h>
```

### Public Member Functions

- GBool **AddStyle** (const char \*pszName, const char \*pszStyleString)  
*Add a new style in the table. No comparison will be done on the Style string, only on the name.*
- GBool **RemoveStyle** (const char \*pszName)  
*Remove a style in the table by its name.*
- GBool **ModifyStyle** (const char \*pszName, const char \*pszStyleString)  
*Modify a style in the table by its name. If the style does not exist, it will be added.*
- GBool **SaveStyleTable** (const char \*pszFilename)  
*Save a style table to a file.*
- GBool **LoadStyleTable** (const char \*pszFilename)

*Load a style table from a file.*

- const char \* **Find** (const char \*pszStyleString)

*Get a style string by name.*

- GBool **IsExist** (const char \*pszName)

*Get the index of a style in the table by its name.*

- const char \* **GetStyleName** (const char \*pszName)

*Get style name by style string.*

- void **Print** (FILE \*fpOut)

*Print a style table to a FILE pointer.*

- void **Clear** ()

*Clear a style table.*

- OGRStyleTable \* **Clone** ()

*Duplicate style table.*

## 12.85.1 Detailed Description

This class represents a style table

## 12.85.2 Member Function Documentation

### 12.85.2.1 GBool OGRStyleTable::AddStyle ( const char \* pszName, const char \* pszStyleString )

Add a new style in the table. No comparison will be done on the Style string, only on the name.

Parameters

<i>pszName</i>	the name the style to add.
<i>pszStyleString</i>	the style string to add.

Returns

TRUE on success, FALSE on error

References [IsExist\(\)](#).

Referenced by [OGRStyleMgr::AddStyle\(\)](#), and [ModifyStyle\(\)](#).

### 12.85.2.2 OGRStyleTable \* OGRStyleTable::Clone ( )

Duplicate style table.

The newly created style table is owned by the caller, and will have it's own reference to the **OGRStyleTable** (p. ??).

Returns

new style table, exactly matching this style table.

References [CSLDuplicate\(\)](#).

Referenced by [OGRLayer::SetStyleTable\(\)](#), and [OGRDataSource::SetStyleTable\(\)](#).

### 12.85.2.3 const char \* OGRStyleTable::Find ( const char \* pszName )

Get a style string by name.

**Parameters**

<i>pszName</i>	the name of the style string to find.
----------------	---------------------------------------

**Returns**

the style string matching the name, NULL if not found or error.

References `IsExist()`.

Referenced by `OGRStyleMgr::GetStyleByName()`.

**12.85.2.4 `const char * OGRStyleTable::GetStyleName ( const char * pszStyleString )`**

Get style name by style string.

**Parameters**

<i>pszStyleString</i>	the style string to look up.
-----------------------	------------------------------

**Returns**

the Name of the matching style string or NULL on error.

References `CSLCount()`.

Referenced by `OGRStyleMgr::GetStyleName()`.

**12.85.2.5 `int OGRStyleTable::IsExist ( const char * pszName )`**

Get the index of a style in the table by its name.

**Parameters**

<i>pszName</i>	the name to look for.
----------------	-----------------------

**Returns**

The index of the style if found, -1 if not found or error.

References `CSLCount()`.

Referenced by `AddStyle()`, `Find()`, and `RemoveStyle()`.

**12.85.2.6 `GBool OGRStyleTable::LoadStyleTable ( const char * pszFilename )`**

Load a style table from a file.

**Parameters**

<i>pszFilename</i>	the name of the file to load from.
--------------------	------------------------------------

**Returns**

TRUE on success, FALSE on error

References `CSLDestroy()`, and `CSLLoad()`.

**12.85.2.7 `GBool OGRStyleTable::ModifyStyle ( const char * pszName, const char * pszStyleString )`**

Modify a style in the table by its name If the style does not exist, it will be added.

## Parameters

<i>pszName</i>	the name of the style to modify.
<i>pszStyleString</i>	the style string.

## Returns

TRUE on success, FALSE on error

References AddStyle(), and RemoveStyle().

12.85.2.8 void OGRStyleTable::Print ( FILE \* *fpOut* )

Print a style table to a FILE pointer.

## Parameters

<i>fpOut</i>	the FILE pointer to print to.
--------------	-------------------------------

12.85.2.9 GBool OGRStyleTable::RemoveStyle ( const char \* *pszName* )

Remove a style in the table by its name.

## Parameters

<i>pszName</i>	the name of the style to remove.
----------------	----------------------------------

## Returns

TRUE on success, FALSE on error

References IsExist().

Referenced by ModifyStyle().

12.85.2.10 GBool OGRStyleTable::SaveStyleTable ( const char \* *pszFilename* )

Save a style table to a file.

## Parameters

<i>pszFilename</i>	the name of the file to save to.
--------------------	----------------------------------

## Returns

TRUE on success, FALSE on error

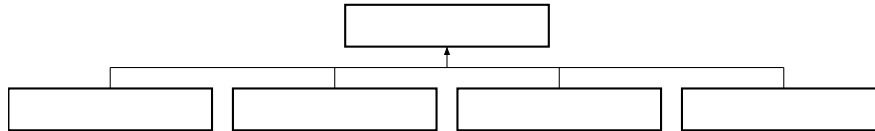
The documentation for this class was generated from the following files:

- **ogr\_featurestyle.h**
- ogrfeaturestyle.cpp

## 12.86 OGRStyleTool Class Reference

```
#include <ogr_featurestyle.h>
```

Inheritance diagram for OGRStyleTool:



### 12.86.1 Detailed Description

This class represents a style tool

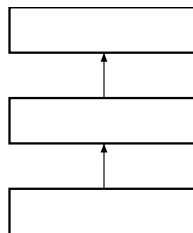
The documentation for this class was generated from the following files:

- **ogr\_featurestyle.h**
- ogrfeaturestyle.cpp

## 12.87 OGRSurface Class Reference

```
#include <ogr_geometry.h>
```

Inheritance diagram for OGRSurface:



### Public Member Functions

- virtual double **get\_Area** () const =0  
*Get the area of the surface object.*
- virtual OGRErr **PointOnSurface** (OGRPoint \*poPoint) const =0  
*This method relates to the SFCOM ISurface::get\_PointOnSurface() method.*

### 12.87.1 Detailed Description

Abstract base class for 2 dimensional objects like polygons.

### 12.87.2 Member Function Documentation

#### 12.87.2.1 double OGRSurface::get\_Area ( ) const [pure virtual]

Get the area of the surface object.

For polygons the area is computed as the area of the outer ring less the area of all internal rings.

This method relates to the SFCOM ISurface::get\_Area() method.



**Returns**

the area of the feature in square units of the spatial reference system in use.

Implemented in **OGRPolygon** (p. ??).

**12.87.2.2 OGRErr OGRSurface::PointOnSurface ( OGRPoint \* poPoint ) const** [pure virtual]

This method relates to the SFCOM ISurface::get\_PointOnSurface() method.

NOTE: Only implemented when GEOS included in build.

**Parameters**

<i>poPoint</i>	point to be set with an internal point.
----------------	---

**Returns**

OGRERR\_NONE if it succeeds or OGRERR\_FAILURE otherwise.

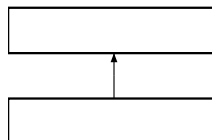
Implemented in **OGRPolygon** (p. ??).

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

- **ogr\_geometry.h**
- **ogrsurface.cpp**

**12.88 OGRUnionLayer Class Reference**

Inheritance diagram for OGRUnionLayer:

**Public Member Functions**

- virtual const char \* **GetName** ()  
*Return the layer name.*
- virtual **OGRwkbGeometryType** **GetGeomType** ()  
*Return the layer geometry type.*
- virtual void **ResetReading** ()  
*Reset feature reading to start on the first feature.*
- virtual **OGRFeature** \* **GetNextFeature** ()  
*Fetch the next available feature from this layer.*
- virtual **OGRFeature** \* **GetFeature** (long nFeatureId)  
*Fetch a feature by its identifier.*
- virtual OGRErr **CreateFeature** (**OGRFeature** \*poFeature)  
*Create and write a new feature within a layer.*
- virtual OGRErr **SetFeature** (**OGRFeature** \*poFeature)  
*Rewrite an existing feature.*
- virtual **OGRFeatureDefn** \* **GetLayerDefn** ()

- Fetch the schema information for this layer.*
- virtual **OGRSpatialReference** \* **GetSpatialRef** ()  
*Fetch the spatial reference system for this layer.*
- virtual int **GetFeatureCount** (int)  
*Fetch the feature count in this layer.*
- virtual OGRErr **SetAttributeFilter** (const char \*)  
*Set a new attribute query.*
- virtual int **TestCapability** (const char \*)  
*Test if this layer supported the named capability.*
- virtual OGRErr **GetExtent** (OGREnvelope \*psExtent, int bForce)  
*Fetch the extent of this layer.*
- virtual void **SetSpatialFilter** (OGRGeometry \*poGeomIn)  
*Set a new spatial filter.*
- virtual OGRErr **SetIgnoredFields** (const char \*\*papszFields)  
*Set which fields can be omitted when retrieving features from the layer.*
- virtual OGRErr **SyncToDisk** ()  
*Flush pending changes to disk.*

## 12.88.1 Member Function Documentation

### 12.88.1.1 OGRErr OGRUnionLayer::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()** (p. ??).

#### Parameters

<i>poFeature</i>	the feature to write to disk.
------------------	-------------------------------

#### Returns

OGRErr\_NONE on success.

Reimplemented from **OGRLayer** (p. ??).

References **CPLError()**, **OGRLayer::CreateFeature()**, **OGRFeature::GetFID()**, **OGRFeature::GetFieldAsString()**, **GetLayerDefn()**, **GetName()**, **OGRFeature::IsFieldSet()**, **OGRFeature::SetFID()**, and **OGRFeature::SetFrom()**.

### 12.88.1.2 OGRErr OGRUnionLayer::GetExtent ( OGREnvelope \* psExtent, int bForce ) [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()** (p. ??) 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()** (p. ??).

## Parameters

<i>psExtent</i>	the structure in which the extent value will be returned.
<i>bForce</i>	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 from **OGRLayer** (p. ??).

### 12.88.1.3 OGRFeature \* OGRUnionLayer::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.

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

Use OGRLayer::TestCapability(OLCRandomRead) to establish if this layer supports efficient random access reading via **GetFeature()** (p. ??); 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 are generally considered interrupted by a **GetFeature()** (p. ??) call.

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

This method is the same as the C function **OGR\_L\_GetFeature()** (p. ??).

## Parameters

<i>nFID</i>	the feature id of the feature to read.
-------------	--

## Returns

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

Reimplemented from **OGRLayer** (p. ??).

References OGRLayer::GetFeature(), and ResetReading().

### 12.88.1.4 int OGRUnionLayer::GetFeatureCount ( int bForce ) [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()** (p. ??).

## Parameters

<i>bForce</i>	Flag indicating whether the count should be computed even if it is expensive.
---------------	---

**Returns**

feature count, -1 if count not known.

Reimplemented from **OGRLayer** (p. ??).

References **OGRLayer::GetFeatureCount()**, **ResetReading()**, and **OGRLayer::SetSpatialFilter()**.

#### 12.88.1.5 **OGRwkbGeometryType** **OGRUnionLayer::GetGeomType** ( ) [virtual]

Return the layer geometry type.

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

This method is the same as the C function **OGR\_L\_GetGeomType()** (p. ??).

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

**Returns**

the geometry type

**Since**

OGR 1.8.0

Reimplemented from **OGRLayer** (p. ??).

References **OGRLayer::GetGeomType()**, and **OGRMergeGeometryTypes()**.

Referenced by **GetLayerDefn()**.

#### 12.88.1.6 **OGRFeatureDefn \*** **OGRUnionLayer::GetLayerDefn** ( ) [virtual]

Fetch the schema information for this layer.

The returned **OGRFeatureDefn** (p. ??) is owned by the **OGRLayer** (p. ??), 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()** (p. ??).

**Returns**

feature definition.

Implements **OGRLayer** (p. ??).

References **OGRFeatureDefn::AddFieldDefn()**, **OGRFeatureDefn::DeleteFieldDefn()**, **OGRFeatureDefn::GetFieldCount()**, **OGRFeatureDefn::GetFieldDefn()**, **OGRFeatureDefn::GetFieldIndex()**, **GetGeomType()**, **OGRLayer::GetLayerDefn()**, **OGRFieldDefn::GetNameRef()**, **OFTString**, **OGRFeatureDefn::Reference()**, and **OGRFeatureDefn::SetGeomType()**.

Referenced by **CreateFeature()**, **GetNextFeature()**, **SetAttributeFilter()**, and **SetFeature()**.

#### 12.88.1.7 **virtual const char\*** **OGRUnionLayer::GetName** ( ) [inline],[virtual]

Return the layer name.

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

This method is the same as the C function **OGR\_L\_GetName()** (p. ??).

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

#### Returns

the layer name (must not be freed)

#### Since

OGR 1.8.0

Reimplemented from **OGRLayer** (p. ??).

Referenced by **CreateFeature()**, and **SetFeature()**.

#### 12.88.1.8 **OGRFeature \* OGRUnionLayer::GetNextFeature** ( ) [virtual]

Fetch the next available feature from this layer.

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

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

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

This method is the same as the C function **OGR\_L\_GetNextFeature()** (p. ??).

#### Returns

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

Implements **OGRLayer** (p. ??).

References **OGRFeature::GetGeometryRef()**, **GetLayerDefn()**, **OGRLayer::GetNextFeature()**, and **ResetReading()**.

#### 12.88.1.9 **OGRSpatialReference \* OGRUnionLayer::GetSpatialRef** ( ) [virtual]

Fetch the spatial reference system for this layer.

The returned object is owned by the **OGRLayer** (p. ??) and should not be modified or freed by the application.

This method is the same as the C function **OGR\_L\_GetSpatialRef()** (p. ??).

#### Returns

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

Reimplemented from **OGRLayer** (p. ??).

References **OGRLayer::GetSpatialRef()**.

#### 12.88.1.10 **void OGRUnionLayer::ResetReading** ( ) [virtual]

Reset feature reading to start on the first feature.

This affects **GetNextFeature()** (p. ??).

This method is the same as the C function **OGR\_L\_ResetReading()** (p. ??).

Implements **OGRLayer** (p. ??).

Referenced by **GetFeature()**, **GetFeatureCount()**, and **GetNextFeature()**.

#### 12.88.1.11 OGRErr OGRUnionLayer::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()** (p. ??) 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()** (p. ??)).

This method is the same as the C function **OGR\_L\_SetAttributeFilter()** (p. ??).

##### Parameters

<i>pszQuery</i>	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 from **OGRLayer** (p. ??).

References **CPLStrdup()**, **GetLayerDefn()**, and **OGRLayer::SetAttributeFilter()**.

#### 12.88.1.12 OGRErr OGRUnionLayer::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** (p. ??).

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

This method is the same as the C function **OGR\_L\_SetFeature()** (p. ??).

##### Parameters

<i>poFeature</i>	the feature to write.
------------------	-----------------------

##### Returns

OGRErr\_NONE if the operation works, otherwise an appropriate error code.

Reimplemented from **OGRLayer** (p. ??).

References **CPLError()**, **OGRFeature::GetFID()**, **OGRFeature::GetFieldAsString()**, **GetLayerDefn()**, **GetName()**, **OGRFeature::IsFieldSet()**, **OGRLayer::SetFeature()**, **OGRFeature::SetFID()**, and **OGRFeature::SetFrom()**.

**12.88.1.13 OGRErr OGRUnionLayer::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()** (p. ??) / **GetNextFeature()** (p. ??) 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()** (p. ??)

**Parameters**

<i>papszFields</i>	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 from **OGRLayer** (p. ??).

References CSLDestroy(), CSLDuplicate(), and OGRLayer::SetIgnoredFields().

**12.88.1.14 void OGRUnionLayer::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()** (p. ??) 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()** (p. ??)) 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()** (p. ??)). In the future this may be generalized.

This method is the same as the C function **OGR\_L\_SetSpatialFilter()** (p. ??).

**Parameters**

<i>poFilter</i>	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 from **OGRLayer** (p. ??).

References OGRLayer::SetSpatialFilter().

**12.88.1.15 OGRErr OGRUnionLayer::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()** (p. ??) that will ensure all data is correctly flushed.

This method is the same as the C function **OGR\_L\_SyncToDisk()** (p. ??).

#### Returns

OGRERR\_NONE if no error occurs (even if nothing is done) or an error code.

Reimplemented from **OGRLayer** (p. ??).

References **OGRLayer::SyncToDisk()**.

**12.88.1.16 int OGRUnionLayer::TestCapability ( const char \* pszCap ) [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()** (p. ??) method is implemented in an optimized way for this layer, as opposed to the default implementation using **ResetReading()** (p. ??) and **GetNextFeature()** (p. ??) to find the requested feature id.
- **OLCSequentialWrite** / "SequentialWrite": TRUE if the **CreateFeature()** (p. ??) method works for this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. ??) class may returned FALSE for other layer instances that are effectively read-only.
- **OLCRandomWrite** / "RandomWrite": TRUE if the **SetFeature()** (p. ??) method is operational on this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. ??) 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** (p. ??) 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()** (p. ??)) 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()** (p. ??)) 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()** (p. ??) call efficiently, otherwise FALSE.
- **OLCCreateField** / "CreateField": TRUE if this layer can create new fields on the current layer using **CreateField()** (p. ??), otherwise FALSE.
- **OLCDeleteField** / "DeleteField": TRUE if this layer can delete existing fields on the current layer using **DeleteField()** (p. ??), otherwise FALSE.
- **OLCReorderFields** / "ReorderFields": TRUE if this layer can reorder existing fields on the current layer using **ReorderField()** (p. ??) or **ReorderFields()** (p. ??), otherwise FALSE.
- **OLCAAlterFieldDefn** / "AlterFieldDefn": TRUE if this layer can alter the definition of an existing field on the current layer using **AlterFieldDefn()** (p. ??), otherwise FALSE.
- **OLCDeleteFeature** / "DeleteFeature": TRUE if the **DeleteFeature()** (p. ??) 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()** (p. ??) method.

This method is the same as the C function **OGR\_L\_TestCapability()** (p. ??).

#### Parameters

<i>pszCap</i>	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.

Implements **OGRLayer** (p. ??).

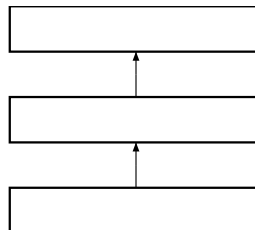
References **OGRLayer::SetSpatialFilter()**.

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

- ogrunionlayer.h
- ogrunionlayer.cpp

## 12.89 OGRWarpedLayer Class Reference

Inheritance diagram for OGRWarpedLayer:



#### Public Member Functions

- 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 **OGRFeature** \* **GetNextFeature** ()  
*Fetch the next available feature from this layer.*
- 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 **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 int **TestCapability** (const char \*)  
*Test if this layer supported the named capability.*

## 12.89.1 Member Function Documentation

### 12.89.1.1 OGRErr OGRWarpedLayer::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()** (p. ??).

#### Parameters

<i>poFeature</i>	the feature to write to disk.
------------------	-------------------------------

#### Returns

OGRERR\_NONE on success.

Reimplemented from **OGRLayerDecorator** (p. ??).

References OGRGeometry::clone(), OGRLayer::CreateFeature(), OGRFeature::GetGeometryRef(), OGRFeature::SetGeometryDirectly(), OGRFeature::StealGeometry(), and OGRGeometry::transform().

### 12.89.1.2 OGRErr OGRWarpedLayer::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()** (p. ??) 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()** (p. ??).

#### Parameters

<i>psExtent</i>	the structure in which the extent value will be returned.
<i>bForce</i>	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 from **OGRLayerDecorator** (p. ??).

References OGRLayer::GetExtent().

**12.89.1.3 OGRFeature \* OGRWarpedLayer::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.

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

Use OGRLayer::TestCapability(OLCRandomRead) to establish if this layer supports efficient random access reading via **GetFeature()** (p. ??); 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 are generally considered interrupted by a **GetFeature()** (p. ??) call.

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

This method is the same as the C function **OGR\_L\_GetFeature()** (p. ??).

**Parameters**

<i>nFID</i>	the feature id of the feature to read.
-------------	--

**Returns**

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

Reimplemented from **OGRLayerDecorator** (p. ??).

References OGRLayer::GetFeature(), OGRFeature::GetGeometryRef(), OGRFeature::StealGeometry(), and OGRGeometry::transform().

**12.89.1.4 int OGRWarpedLayer::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()** (p. ??).

**Parameters**

<i>bForce</i>	Flag indicating whether the count should be computed even if it is expensive.
---------------	---

**Returns**

feature count, -1 if count not known.

Reimplemented from **OGRLayerDecorator** (p. ??).

References **OGRLayer::GetFeatureCount()**.

#### 12.89.1.5 **OGRFeature \* OGRWarpedLayer::GetNextFeature ( )** [virtual]

Fetch the next available feature from this layer.

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

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

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

This method is the same as the C function **OGR\_L\_GetNextFeature()** (p. ??).

**Returns**

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

Reimplemented from **OGRLayerDecorator** (p. ??).

References **OGRFeature::GetGeometryRef()**, **OGRLayer::GetNextFeature()**, **OGRFeature::StealGeometry()**, and **OGRGeometry::transform()**.

#### 12.89.1.6 **OGRSpatialReference \* OGRWarpedLayer::GetSpatialRef ( )** [virtual]

Fetch the spatial reference system for this layer.

The returned object is owned by the **OGRLayer** (p. ??) and should not be modified or freed by the application.

This method is the same as the C function **OGR\_L\_GetSpatialRef()** (p. ??).

**Returns**

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

Reimplemented from **OGRLayerDecorator** (p. ??).

#### 12.89.1.7 **OGRERR OGRWarpedLayer::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** (p. ??).

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

This method is the same as the C function **OGR\_L\_SetFeature()** (p. ??).

## Parameters

<i>poFeature</i>	the feature to write.
------------------	-----------------------

## Returns

OGRERR\_NONE if the operation works, otherwise an appropriate error code.

Reimplemented from **OGRLayerDecorator** (p. ??).

References OGRGeometry::clone(), OGRFeature::GetGeometryRef(), OGRLayer::SetFeature(), OGRFeature::SetGeometryDirectly(), OGRFeature::StealGeometry(), and OGRGeometry::transform().

#### 12.89.1.8 void OGRWarpedLayer::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()** (p. ??) 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()** (p. ??)) 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()** (p. ??)). In the future this may be generalized.

This method is the same as the C function **OGR\_L\_SetSpatialFilter()** (p. ??).

## Parameters

<i>poFilter</i>	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 from **OGRLayerDecorator** (p. ??).

References OGRGeometry::getEnvelope(), OGRLayer::SetSpatialFilter(), and OGRLayer::SetSpatialFilterRect().

#### 12.89.1.9 void OGRWarpedLayer::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()** (p. ??) 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()** (p. ??)). Internally this method is normally implemented as creating a 5 vertex closed rectangular polygon and passing it to **OGRLayer::SetSpatialFilter()** (p. ??). 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()** (p. ??).

## Parameters

<i>dfMinX</i>	the minimum X coordinate for the rectangular region.
---------------	--

<i>dfMinY</i>	the minimum Y coordinate for the rectangular region.
<i>dfMaxX</i>	the maximum X coordinate for the rectangular region.
<i>dfMaxY</i>	the maximum Y coordinate for the rectangular region.

Reimplemented from **OGRLayerDecorator** (p. ??).

References OGRLayer::SetSpatialFilterRect().

#### 12.89.1.10 int OGRWarpedLayer::TestCapability ( const char \* pszCap ) [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()** (p. ??) method is implemented in an optimized way for this layer, as opposed to the default implementation using **ResetReading()** (p. ??) and **GetNextFeature()** (p. ??) to find the requested feature id.
- **OLCSequentialWrite** / "SequentialWrite": TRUE if the **CreateFeature()** (p. ??) method works for this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. ??) class may returned FALSE for other layer instances that are effectively read-only.
- **OLCRandomWrite** / "RandomWrite": TRUE if the **SetFeature()** (p. ??) method is operational on this layer. Note this means that this particular layer is writable. The same **OGRLayer** (p. ??) 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** (p. ??) 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()** (p. ??)) 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()** (p. ??)) 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()** (p. ??) call efficiently, otherwise FALSE.
- **OLCCreateField** / "CreateField": TRUE if this layer can create new fields on the current layer using **CreateField()** (p. ??), otherwise FALSE.
- **OLCDeleteField** / "DeleteField": TRUE if this layer can delete existing fields on the current layer using **DeleteField()** (p. ??), otherwise FALSE.
- **OLCReorderFields** / "ReorderFields": TRUE if this layer can reorder existing fields on the current layer using **ReorderField()** (p. ??) or **ReorderFields()** (p. ??), otherwise FALSE.
- **OLCAlterFieldDefn** / "AlterFieldDefn": TRUE if this layer can alter the definition of an existing field on the current layer using **AlterFieldDefn()** (p. ??), otherwise FALSE.
- **OLCDeleteFeature** / "DeleteFeature": TRUE if the **DeleteFeature()** (p. ??) 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()** (p. ??) method.

This method is the same as the C function **OGR\_L\_TestCapability()** (p. ??).

#### Parameters

<i>pszCap</i>	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.

Reimplemented from **OGRLayerDecorator** (p. ??).

References `OGRLayer::TestCapability()`.

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

- ogrwarpedlayer.h
- ogrwarpedlayer.cpp

## 12.90 ParseContext Struct Reference

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

- cpl\_minixml.cpp

## 12.91 PCIDatums Struct Reference

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

- ogr\_srs\_pci.cpp

## 12.92 projUV Struct Reference

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

- ogrct.cpp

## 12.93 RingBuffer Class Reference

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

- cpl\_vsil\_curl\_streaming.cpp

## 12.94 SFRegion Class Reference

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

- cpl\_vsil\_sparsefile.cpp

## 12.95 StackContext Struct Reference

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

- `cpl_minixml.cpp`

## 12.96 swq\_col\_def Struct Reference

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

- `swq.h`

## 12.97 swq\_expr\_node Class Reference

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

- `swq.h`
- `swq_expr_node.cpp`

## 12.98 swq\_field\_list Class Reference

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

- `swq.h`

## 12.99 swq\_join\_def Struct Reference

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

- `swq.h`

## 12.100 swq\_op\_registrar Class Reference

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

- `swq.h`
- `swq_op_registrar.cpp`

## 12.101 swq\_operation Class Reference

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

- `swq.h`



## 12.102 swq\_order\_def Struct Reference

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

- swq.h

## 12.103 swq\_parse\_context Class Reference

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

- swq.h

## 12.104 swq\_select Class Reference

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

- swq.h
- swq\_select.cpp

## 12.105 swq\_summary Struct Reference

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

- swq.h

## 12.106 swq\_table\_def Struct Reference

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

- swq.h

## 12.107 tm\_unz\_s Struct Reference

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

- cpl\_minizip\_unzip.h

## 12.108 tm\_zip\_s Struct Reference

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

- cpl\_minizip\_zip.h

## 12.109 unz\_file\_info\_internal\_s Struct Reference

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

- cpl\_minizip\_unzip.cpp

### 12.110 unz\_file\_info\_s Struct Reference

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

- cpl\_minizip\_unzip.h

### 12.111 unz\_file\_pos\_s Struct Reference

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

- cpl\_minizip\_unzip.h

### 12.112 unz\_global\_info\_s Struct Reference

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

- cpl\_minizip\_unzip.h

### 12.113 unz\_s Struct Reference

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

- cpl\_minizip\_unzip.cpp

### 12.114 VSIArchiveContent Struct Reference

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

- cpl\_vsi\_virtual.h

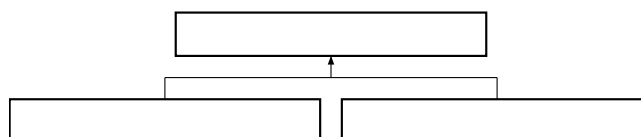
### 12.115 VSIArchiveEntry Struct Reference

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

- cpl\_vsi\_virtual.h

### 12.116 VSIArchiveEntryFileOffset Class Reference

Inheritance diagram for VSIArchiveEntryFileOffset:

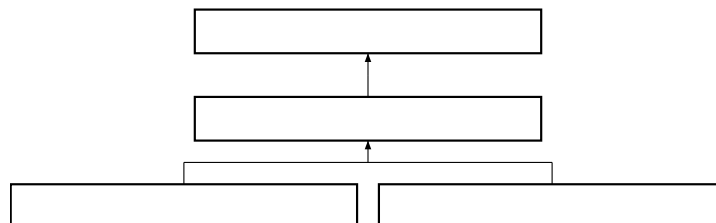


The documentation for this class was generated from the following files:

- `cpl_vsi_virtual.h`
- `cpl_vsil_abstract_archive.cpp`

## 12.117 VSIArchiveFilesystemHandler Class Reference

Inheritance diagram for VSIArchiveFilesystemHandler:

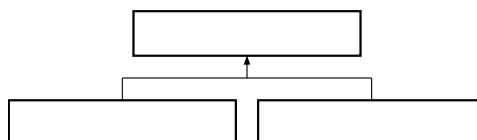


The documentation for this class was generated from the following files:

- `cpl_vsi_virtual.h`
- `cpl_vsil_abstract_archive.cpp`

## 12.118 VSIArchiveReader Class Reference

Inheritance diagram for VSIArchiveReader:

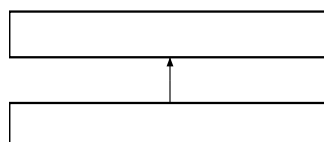


The documentation for this class was generated from the following files:

- `cpl_vsi_virtual.h`
- `cpl_vsil_abstract_archive.cpp`

## 12.119 VSIBufferedReaderHandle Class Reference

Inheritance diagram for VSIBufferedReaderHandle:



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

- `cpl_vsil_buffered_reader.cpp`

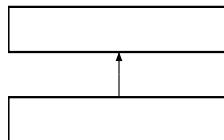
## 12.120 VSICacheChunk Class Reference

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

- `cpl_vsil_cache.cpp`

## 12.121 VSICachedFile Class Reference

Inheritance diagram for VSICachedFile:

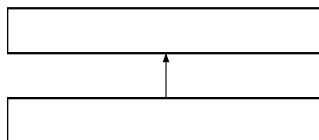


The documentation for this class was generated from the following file:

- `cpl_vsil_cache.cpp`

## 12.122 VSICurlFilesystemHandler Class Reference

Inheritance diagram for VSICurlFilesystemHandler:

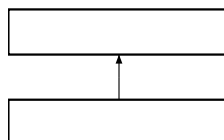


The documentation for this class was generated from the following file:

- `cpl_vsil_curl.cpp`

## 12.123 VSICurlHandle Class Reference

Inheritance diagram for VSICurlHandle:

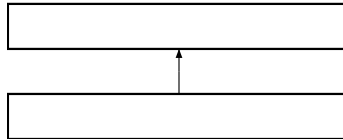


The documentation for this class was generated from the following file:

- `cpl_vsil_curl.cpp`

## 12.124 VSICurlStreamingFSHandler Class Reference

Inheritance diagram for VSICurlStreamingFSHandler:

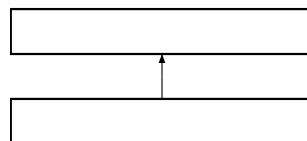


The documentation for this class was generated from the following file:

- `cpl_vsil_curl_streaming.cpp`

## 12.125 VSICurlStreamingHandle Class Reference

Inheritance diagram for VSICurlStreamingHandle:



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

- `cpl_vsil_curl_streaming.cpp`

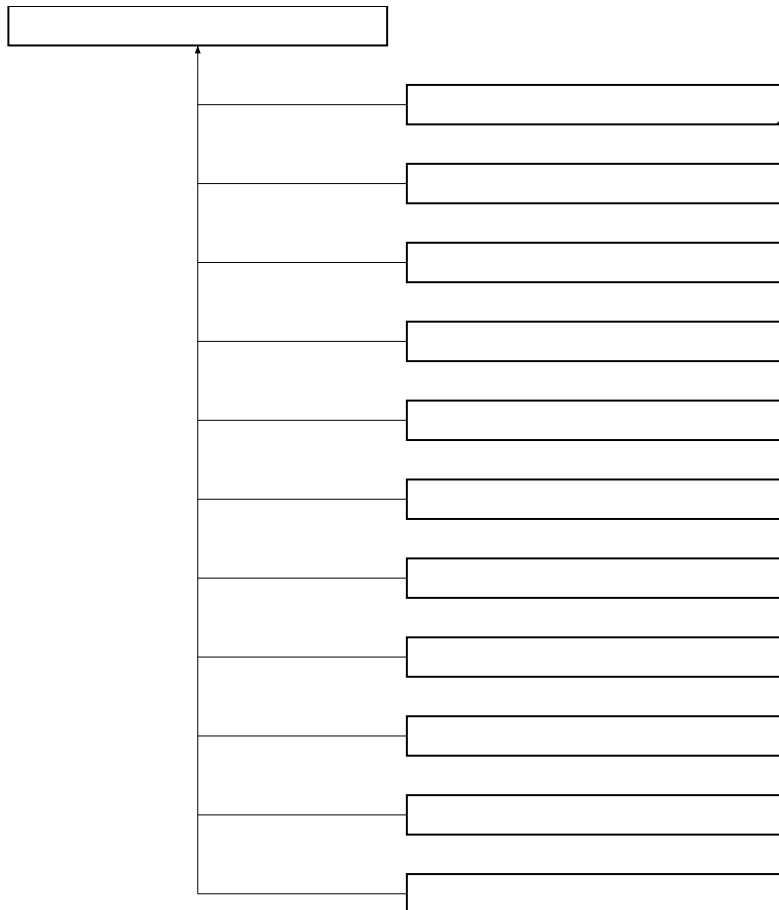
## 12.126 VSIFileManager Class Reference

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

- `cpl_vsi_virtual.h`
- `cpl_vsil.cpp`

## 12.127 VSIFilesystemHandler Class Reference

Inheritance diagram for VSIFilesystemHandler:

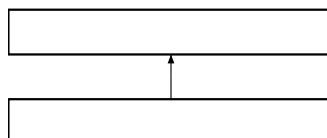


The documentation for this class was generated from the following file:

- `cpl_vsi_virtual.h`

## 12.128 VSIGZipFilesystemHandler Class Reference

Inheritance diagram for VSIGZipFilesystemHandler:

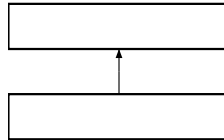


The documentation for this class was generated from the following file:

- `cpl_vsil_gzip.cpp`

## 12.129 VSIGZipHandle Class Reference

Inheritance diagram for VSIGZipHandle:

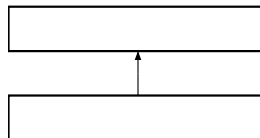


The documentation for this class was generated from the following file:

- `cpl_vsil_gzip.cpp`

## 12.130 VSIGZipWriteHandle Class Reference

Inheritance diagram for VSIGZipWriteHandle:



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

- `cpl_vsil_gzip.cpp`

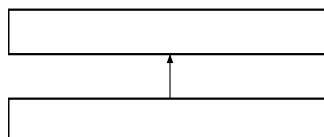
## 12.131 VSIMemFile Class Reference

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

- `cpl_vsi_mem.cpp`

## 12.132 VSIMemFilesystemHandler Class Reference

Inheritance diagram for VSIMemFilesystemHandler:

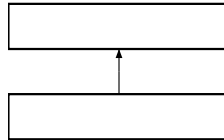


The documentation for this class was generated from the following file:

- `cpl_vsi_mem.cpp`

## 12.133 VSIMemHandle Class Reference

Inheritance diagram for VSIMemHandle:



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

- `cpl_vsi_mem.cpp`

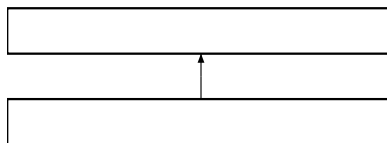
### 12.134 VSIReadDirRecursiveTask Struct Reference

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

- `cpl_vsil.cpp`

### 12.135 VSISparseFileFilesystemHandler Class Reference

Inheritance diagram for VSISparseFileFilesystemHandler:

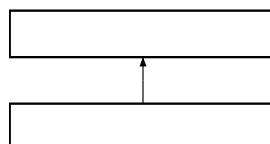


The documentation for this class was generated from the following file:

- `cpl_vsil_sparsefile.cpp`

### 12.136 VSISparseFileHandle Class Reference

Inheritance diagram for VSISparseFileHandle:



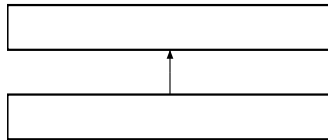
The documentation for this class was generated from the following file:

- `cpl_vsil_sparsefile.cpp`

### 12.137 VSIStdinFilesystemHandler Class Reference

Inheritance diagram for VSIStdinFilesystemHandler:



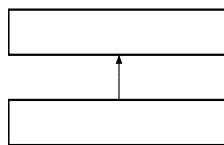


The documentation for this class was generated from the following file:

- `cpl_vsil_stdin.cpp`

## 12.138 VSIStdinHandle Class Reference

Inheritance diagram for VSIStdinHandle:

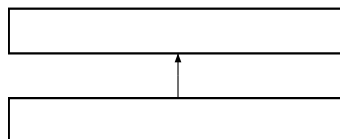


The documentation for this class was generated from the following file:

- `cpl_vsil_stdin.cpp`

## 12.139 VSIStdoutFilesystemHandler Class Reference

Inheritance diagram for VSIStdoutFilesystemHandler:

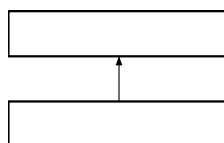


The documentation for this class was generated from the following file:

- `cpl_vsil_stdout.cpp`

## 12.140 VSIStdoutHandle Class Reference

Inheritance diagram for VSIStdoutHandle:

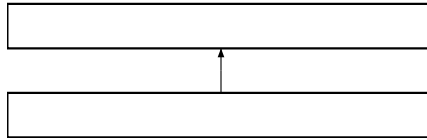


The documentation for this class was generated from the following file:

- `cpl_vsil_stdout.cpp`

### 12.141 VSIStdoutRedirectFilesystemHandler Class Reference

Inheritance diagram for VSIStdoutRedirectFilesystemHandler:

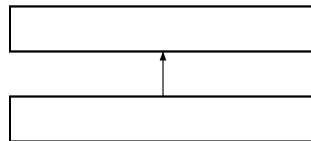


The documentation for this class was generated from the following file:

- `cpl_vsil_stdout.cpp`

### 12.142 VSIStdoutRedirectHandle Class Reference

Inheritance diagram for VSIStdoutRedirectHandle:

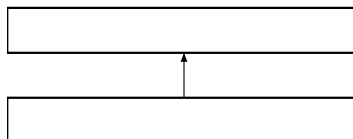


The documentation for this class was generated from the following file:

- `cpl_vsil_stdout.cpp`

### 12.143 VSISubFileFilesystemHandler Class Reference

Inheritance diagram for VSISubFileFilesystemHandler:

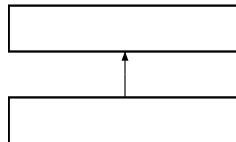


The documentation for this class was generated from the following file:

- `cpl_vsil_subfile.cpp`

### 12.144 VSISubFileHandle Class Reference

Inheritance diagram for VSISubFileHandle:

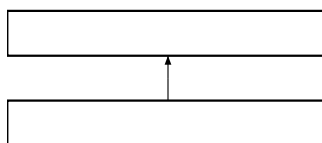


The documentation for this class was generated from the following file:

- `cpl_vsil_subfile.cpp`

## 12.145 VSITarEntryFileOffset Class Reference

Inheritance diagram for VSITarEntryFileOffset:

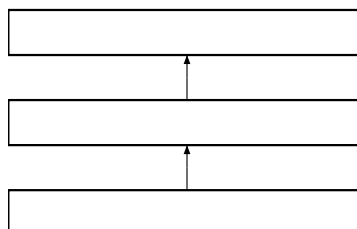


The documentation for this class was generated from the following file:

- `cpl_vsil_tar.cpp`

## 12.146 VSITarFilesystemHandler Class Reference

Inheritance diagram for VSITarFilesystemHandler:

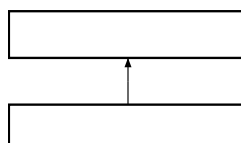


The documentation for this class was generated from the following file:

- `cpl_vsil_tar.cpp`

## 12.147 VSITarReader Class Reference

Inheritance diagram for VSITarReader:

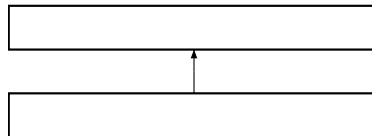


The documentation for this class was generated from the following file:

- `cpl_vsil_tar.cpp`

## 12.148 VSIUnixStdioFilesystemHandler Class Reference

Inheritance diagram for VSIUnixStdioFilesystemHandler:

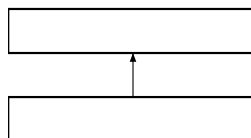


The documentation for this class was generated from the following file:

- `cpl_vsil_unix_stdio_64.cpp`

## 12.149 VSIUnixStdioHandle Class Reference

Inheritance diagram for VSIUnixStdioHandle:

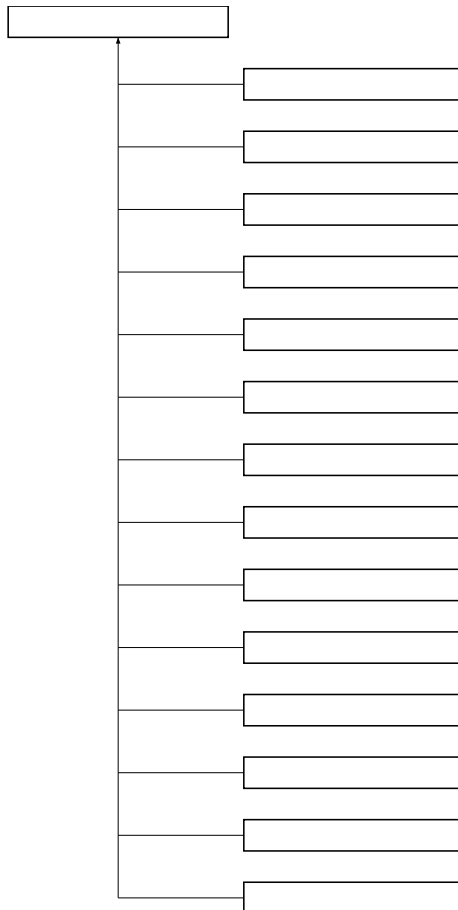


The documentation for this class was generated from the following file:

- `cpl_vsil_unix_stdio_64.cpp`

## 12.150 VSIVirtualHandle Class Reference

Inheritance diagram for VSIVirtualHandle:

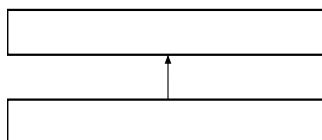


The documentation for this class was generated from the following files:

- cpl\_vsi\_virtual.h
- cpl\_vsil.cpp

## 12.151 VSZipEntryFileOffset Class Reference

Inheritance diagram for VSZipEntryFileOffset:

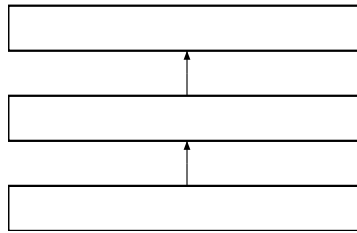


The documentation for this class was generated from the following file:

- cpl\_vsil\_gzip.cpp

## 12.152 VSZipFilesystemHandler Class Reference

Inheritance diagram for VSZipFilesystemHandler:

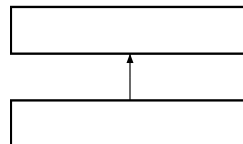


The documentation for this class was generated from the following file:

- `cpl_vsil_gzip.cpp`

### 12.153 VSZipReader Class Reference

Inheritance diagram for VSZipReader:

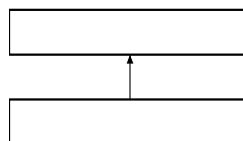


The documentation for this class was generated from the following file:

- `cpl_vsil_gzip.cpp`

### 12.154 VSZipWriteHandle Class Reference

Inheritance diagram for VSZipWriteHandle:



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

- `cpl_vsil_gzip.cpp`

### 12.155 WriteFuncStruct Struct Reference

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

- `cpl_vsil_curl.cpp`
- `cpl_vsil_curl_streaming.cpp`

## 12.156 yyalloc Union Reference

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

- swq\_parser.cpp

## 12.157 zip\_fileinfo Struct Reference

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

- cpl\_minizip\_zip.h

## 12.158 zip\_internal Struct Reference

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

- cpl\_minizip\_zip.cpp

## 12.159 zlib\_filefunc\_def\_s Struct Reference

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

- cpl\_minizip\_ioapi.h





# Chapter 13

## File Documentation

### 13.1 cpl\_conv.h File Reference

```
#include "cpl_port.h"  
#include "cpl_vsi.h"  
#include "cpl_error.h"
```

#### Classes

- struct **CPLSharedFileInfo**
- class **CPLLocaleC**

#### Functions

- const char \* **CPLGetConfigOption** (const char \*, const char \*)
- void **CPLSetConfigOption** (const char \*, const char \*)
- void **CPLSetThreadLocalConfigOption** (const char \*pszKey, const char \*pszValue)
- void \* **CPLMalloc** (size\_t)
- void \* **CPLCalloc** (size\_t, size\_t)
- void \* **CPLRealloc** (void \*, size\_t)
- char \* **CPLStrdup** (const char \*)
- char \* **CPLStrlwr** (char \*)
- char \* **CPLFGets** (char \*, int, FILE \*)
- const char \* **CPLReadLine** (FILE \*)
- const char \* **CPLReadLineL** (VSILFILE \*)
- const char \* **CPLReadLine2L** (VSILFILE \*, int nMaxCols, char \*\*papszOptions)
- double **CPLAtof** (const char \*)
- double **CPLAtofDelim** (const char \*, char)
- double **CPLStrtod** (const char \*, char \*\*)
- double **CPLStrtodDelim** (const char \*, char \*\*, char)
- float **CPLStrtof** (const char \*, char \*\*)
- float **CPLStrtofDelim** (const char \*, char \*\*, char)
- double **CPLAtofM** (const char \*)
- char \* **CPLScanString** (const char \*, int, int, int)
- double **CPLScanDouble** (const char \*, int)
- long **CPLScanLong** (const char \*, int)
- unsigned long **CPLScanULong** (const char \*, int)
- GUIntBig **CPLScanUIntBig** (const char \*, int)

- void \* **CPLScanPointer** (const char \*, int)
- int **CPLPrintString** (char \*, const char \*, int)
- int **CPLPrintStringFill** (char \*, const char \*, int)
- int **CPLPrintInt32** (char \*, GLint32, int)
- int **CPLPrintUIntBig** (char \*, GUIntBig, int)
- int **CPLPrintDouble** (char \*, const char \*, double, const char \*)
- int **CPLPrintTime** (char \*, int, const char \*, const struct tm \*, const char \*)
- int **CPLPrintPointer** (char \*, void \*, int)
- void \* **CPLGetSymbol** (const char \*, const char \*)
- int **CPLGetExecPath** (char \*pszPathBuf, int nMaxLength)
- const char \* **CPLGetPath** (const char \*)
- const char \* **CPLGetDirname** (const char \*)
- const char \* **CPLGetFilename** (const char \*)
- const char \* **CPLGetBasename** (const char \*)
- const char \* **CPLGetExtension** (const char \*)
- char \* **CPLGetCurrentDir** (void)
- const char \* **CPLFormFilename** (const char \*pszPath, const char \*pszBasename, const char \*psz-Extension)
- const char \* **CPLFormCIFilename** (const char \*pszPath, const char \*pszBasename, const char \*psz-Extension)
- const char \* **CPLResetExtension** (const char \*, const char \*)
- const char \* **CPLProjectRelativeFilename** (const char \*pszProjectDir, const char \*pszSecondaryFilename)
- int **CPLIsFilenameRelative** (const char \*pszFilename)
- const char \* **CPLExtractRelativePath** (const char \*, const char \*, int \*)
- const char \* **CPLCleanTrailingSlash** (const char \*)
- char \*\* **CPLCorrespondingPaths** (const char \*pszOldFilename, const char \*pszNewFilename, char \*\*papszFileList)
- int **CPLCheckForFile** (char \*pszFilename, char \*\*papszSiblingList)
- const char \* **CPLGenerateTempFilename** (const char \*pszStem)
- FILE \* **CPLOpenShared** (const char \*, const char \*, int)
- void **CPLCloseShared** (FILE \*)
- **CPLSharedFileInfo** \* **CPLGetSharedList** (int \*)
- void **CPLDumpSharedList** (FILE \*)
- double **CPLPackedDMSToDec** (double)
- double **CPLDecToPackedDMS** (double dfDec)
- int **CPLUnlinkTree** (const char \*)
- void \* **CPLZLibDeflate** (const void \*ptr, size\_t nBytes, int nLevel, void \*outptr, size\_t nOutAvailableBytes, size\_t \*pnOutBytes)  
*Compress a buffer with ZLib DEFLATE compression.*
- void \* **CPLZLibInflate** (const void \*ptr, size\_t nBytes, void \*outptr, size\_t nOutAvailableBytes, size\_t \*pnOutBytes)  
*Uncompress a buffer compressed with ZLib DEFLATE compression.*

### 13.1.1 Detailed Description

Various convenience functions for CPL.

## 13.1.2 Function Documentation

### 13.1.2.1 double CPLAtof ( const char \* *nptr* )

Converts ASCII string to floating point number.

This function converts the initial portion of the string pointed to by *nptr* to double floating point representation. The behaviour is the same as

```
CPLStrtod(nptr, (char **)NULL);
```

This function does the same as standard `atof(3)`, but does not take locale in account. That means, the decimal delimiter is always `'.'` (decimal point). Use **CPLAtofDelim()** (p. ??) function if you want to specify custom delimiter.

**IMPORTANT NOTE.** Existence of this function does not mean you should always use it. Sometimes you should use standard locale aware `atof(3)` and its family. When you need to process the user's input (for example, command line parameters) use `atof(3)`, because user works in localized environment and her input will be done accordingly the locale set. In particular that means we should not make assumptions about character used as decimal delimiter, it can be either `"."` or `">".` But when you are parsing some ASCII file in predefined format, you most likely need **CPLAtof()** (p. ??), because such files distributed across the systems with different locales and floating point representation should be considered as a part of file format. If the format uses `"."` as a delimiter the same character must be used when parsing number regardless of actual locale setting.

#### Parameters

<i>nptr</i>	Pointer to string to convert.
-------------	-------------------------------

#### Returns

Converted value, if any.

References `CPLStrtod()`.

Referenced by `OGRSpatialReference::CloneGeogCS()`, `CPLScanDouble()`, `OGRSpatialReference::exportToPCI()`, `OGRSpatialReference::exportToProj4()`, `OGRSpatialReference::Fixup()`, `OGRSpatialReference::GetAngularUnits()`, `OGRSpatialReference::GetInvFlattening()`, `OGRSpatialReference::GetPrimeMeridian()`, `OGRSpatialReference::GetProjParm()`, `OGRSpatialReference::GetSemiMajor()`, `OGRSpatialReference::GetTargetLinearUnits()`, `OGRSpatialReference::GetTOWGS84()`, `OGRSpatialReference::importFromOzi()`, `OGRSpatialReference::importFromPCI()`, `OGRSpatialReference::importFromProj4()`, `OGRSpatialReference::importFromWMSAUTO()`, `OGRSpatialReference::IsSameGeogCS()`, `OGRSpatialReference::IsSameVertCS()`, `OGRSpatialReference::SetGeogCS()`, and `OGRSpatialReference::SetStatePlane()`.

### 13.1.2.2 double CPLAtofDelim ( const char \* *nptr*, char *point* )

Converts ASCII string to floating point number.

This function converts the initial portion of the string pointed to by *nptr* to double floating point representation. The behaviour is the same as

```
CPLStrtodDelim(nptr, (char **)NULL, point);
```

This function does the same as standard `atof(3)`, but does not take locale in account. Instead of locale defined decimal delimiter you can specify your own one. Also see notes for **CPLAtof()** (p. ??) function.

#### Parameters

<i>nptr</i>	Pointer to string to convert.
<i>point</i>	Decimal delimiter.

#### Returns

Converted value, if any.

References `CPLStrtodDelim()`.

### 13.1.2.3 double CPLAtofM ( const char \* *nptr* )

Converts ASCII string to floating point number using any numeric locale.

This function converts the initial portion of the string pointed to by *nptr* to double floating point representation. This function does the same as standard `atof()`, but it allows a variety of locale representations. That is it supports numeric values with either a comma or a period for the decimal delimiter.

PS. The M stands for Multi-lingual.

#### Parameters

<i>nptr</i>	The string to convert.
-------------	------------------------

#### Returns

Converted value, if any. Zero on failure.

References `CPLStrtodDelim()`.

Referenced by `OGRSpatialReference::importFromOzi()`, `OGRSpatialReference::importFromProj4()`, and `OGRGeocodeCreateSession()`.

### 13.1.2.4 void\* CPLCalloc ( size\_t *nCount*, size\_t *nSize* )

Safe version of `calloc()`.

This function is like the C library `calloc()`, but raises a `CE_Fatal` error with **CPLError()** (p. ??) if it fails to allocate the desired memory. It should be used for small memory allocations that are unlikely to fail and for which the application is unwilling to test for out of memory conditions. It uses `VSIAlloc()` to get the memory, so any hooking of `VSI-Calloc()` will apply to **CPLCalloc()** (p. ??) as well. `CPLFree()` or `VSIFree()` can be used free memory allocated by **CPLCalloc()** (p. ??).

#### Parameters

<i>nCount</i>	number of objects to allocate.
<i>nSize</i>	size (in bytes) of object to allocate.

#### Returns

pointer to newly allocated memory, only NULL if `nSize * nCount` is NULL.

References `CPLMalloc()`.

Referenced by `OGRSFDriverRegistrar::AutoLoadDrivers()`, `OGRDataSource::CopyLayer()`, `CPLCreateXMLNode()`, `CPLHashSetNew()`, `CPLHTTPFetch()`, `CSLTokenizeString2()`, `OGR_SRSNode::exportToWkt()`, `OGRPolygon::exportToWkt()`, `OGRGeometryCollection::exportToWkt()`, `OGRMultiPolygon::exportToWkt()`, `OGRMultiLineString::exportToWkt()`, `OGR_SRSNode::FixupOrdering()`, `CPLDBCStatement::GetColumns()`, `OGRBuildPolygonFromEdges()`, `OGRGeocodeCreateSession()`, and `OPTGetParameterList()`.

### 13.1.2.5 int CPLCheckForFile ( char \* *pszFilename*, char \*\* *papszSiblingFiles* )

Check for file existence.

The function checks if a named file exists in the filesystem, hopefully in an efficient fashion if a sibling file list is available. It exists primarily to do faster file checking for functions like GDAL open methods that get a list of files from the target directory.

If the sibling file list exists (is not NULL) it is assumed to be a list of files in the same directory as the target file, and it will be checked (case insensitively) for a match. If a match is found, *pszFilename* is updated with the correct case and TRUE is returned.

If papszSiblingFiles is NULL, a **VSISStatL()** (p. ??) is used to test for the files existence, and no case insensitive testing is done.

## Parameters

<i>pszFilename</i>	name of file to check for - filename case updated in some cases.
<i>papszSibling-Files</i>	a list of files in the same directory as pszFilename if available, or NULL. This list should have no path components.

## Returns

TRUE if a match is found, or FALSE if not.

References CPLGetFilename(), and VSISatL().

13.1.2.6 `const char* CPLCleanTrailingSlash ( const char * pszPath )`

Remove trailing forward/backward slash from the path for unix/windows resp.

Returns a string containing the portion of the passed path string with trailing slash removed. If there is no path in the passed filename an empty string will be returned (not NULL).

```
CPLCleanTrailingSlash( "abc/def/" ) == "abc/def"
CPLCleanTrailingSlash( "abc/def" ) == "abc/def"
CPLCleanTrailingSlash( "c:\abc\def\" ) == "c:\abc\def"
CPLCleanTrailingSlash( "c:\abc\def" ) == "c:\abc\def"
CPLCleanTrailingSlash( "abc" ) == "abc"
```

## Parameters

<i>pszPath</i>	the path to be cleaned up
----------------	---------------------------

## Returns

Path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call.

References CPLStrlcpy().

13.1.2.7 `void CPLCloseShared ( FILE * fp )`

Close shared file.

Dereferences the indicated file handle, and closes it if the reference count has dropped to zero. A **CPL**Error() (p. ??) is issued if the file is not in the shared file list.

## Parameters

<i>fp</i>	file handle from <b>CPL</b> OpenShared() (p. ??) to deaccess.
-----------	---

References CPLError(), and VSIFCloseL().

13.1.2.8 `char** CPLCorrespondingPaths ( const char * pszOldFilename, const char * pszNewFilename, char ** papszFileList )`

Identify corresponding paths.

Given a prototype old and new filename this function will attempt to determine corresponding names for a set of other old filenames that will rename them in a similar manner. This correspondance assumes there are two possibly kinds of renaming going on. A change of path, and a change of filename stem.

If a consistent renaming cannot be established for all the files this function will return indicating an error.

The returned file list becomes owned by the caller and should be destroyed with **CSL**Destroy() (p. ??).

## Parameters

<i>pszOldFilename</i>	path to old prototype file.
<i>pszNew-Filename</i>	path to new prototype file.
<i>papszFileList</i>	list of other files associated with pszOldFilename to rename similarly.

## Returns

a list of files corresponding to papszFileList but renamed to correspond to pszNewFilename.

References CPLError(), CPLFormFilename(), CPLGetBasename(), CPLGetFilename(), CPLGetPath(), and CSL-Count().

13.1.2.9 double CPLDecToPackedDMS ( double *dfDec* )

Convert decimal degrees into packed DMS value (DDDMMMSSS.SS).

This function converts a value, specified in decimal degrees into packed DMS angle. The standard packed DMS format is:

degrees \* 1000000 + minutes \* 1000 + seconds

See also **CPLPackedDMSToDec()** (p. ??).

## Parameters

<i>dfDec</i>	Angle in decimal degrees.
--------------	---------------------------

## Returns

Angle in packed DMS format.

Referenced by OGRSpatialReference::exportToUSGS().

13.1.2.10 void CPLDumpSharedList ( FILE \* *fp* )

Report open shared files.

Dumps all open shared files to the indicated file handle. If the file handle is NULL information is sent via the **CPLDebug()** (p. ??) call.

## Parameters

<i>fp</i>	File handle to write to.
-----------	--------------------------

References CPLDebug().

13.1.2.11 const char\* CPLExtractRelativePath ( const char \* *pszBaseDir*, const char \* *pszTarget*, int \* *pbGotRelative* )

Get relative path from directory to target file.

Computes a relative path for pszTarget relative to pszBaseDir. Currently this only works if they share a common base path. The returned path is normally into the pszTarget string. It should only be considered valid as long as pszTarget is valid or till the next call to this function, whichever comes first.

## Parameters

<i>pszBaseDir</i>	the name of the directory relative to which the path should be computed. pszBaseDir may be NULL in which case the original target is returned without relativizing.
<i>pszTarget</i>	the filename to be changed to be relative to pszBaseDir.
<i>pbGotRelative</i>	Pointer to location in which a flag is placed indicating that the returned path is relative to the basename (TRUE) or not (FALSE). This pointer may be NULL if flag is not desired.

**Returns**

an adjusted path or the original if it could not be made relative to the pszBaseFile's path.

References CPLIsFilenameRelative().

### 13.1.2.12 char\* CPLFgets ( char \* pszBuffer, int nBufferSize, FILE \* fp )

Reads in at most one less than nBufferSize characters from the fp stream and stores them into the buffer pointed to by pszBuffer. Reading stops after an EOF or a newline. If a newline is read, it is *not* stored into the buffer. A '\0' is stored after the last character in the buffer. All three types of newline terminators recognized by the **CPLFgets()** (p. ??): single '\r' and '\n' and '\r\n' combination.

**Parameters**

<i>pszBuffer</i>	pointer to the targeting character buffer.
<i>nBufferSize</i>	maximum size of the string to read (not including terminating '\0').
<i>fp</i>	file pointer to read from.

**Returns**

pointer to the pszBuffer containing a string read from the file or NULL if the error or end of file was encountered.

References CPLDebug().

Referenced by CPLReadLine().

### 13.1.2.13 const char\* CPLFormCIFilename ( const char \* pszPath, const char \* pszBasename, const char \* pszExtension )

Case insensitive file searching, returning full path.

This function tries to return the path to a file regardless of whether the file exactly matches the basename, and extension case, or is all upper case, or all lower case. The path is treated as case sensitive. This function is equivalent to **CPLFormFilename()** (p. ??) on case insensitive file systems (like Windows).

**Parameters**

<i>pszPath</i>	directory path to the directory containing the file. This may be relative or absolute, and may have a trailing path separator or not. May be NULL.
<i>pszBasename</i>	file basename. May optionally have path and/or extension. May not be NULL.
<i>pszExtension</i>	file extension, optionally including the period. May be NULL.

**Returns**

a fully formed filename in an internal static string. Do not modify or free the returned string. The string may be destroyed by the next CPL call.

References CPLFormFilename(), CPLMalloc(), VSIsCaseSensitiveFS(), and VSStatExL().

### 13.1.2.14 const char\* CPLFormFilename ( const char \* pszPath, const char \* pszBasename, const char \* pszExtension )

Build a full file path from a passed path, file basename and extension.

The path, and extension are optional. The basename may in fact contain an extension if desired.



```

CPLFormFilename("abc/xyz","def", ".dat" ) == "abc/xyz/def.dat"
CPLFormFilename(NULL,"def", NULL ) == "def"
CPLFormFilename(NULL,"abc/def.dat", NULL ) == "abc/def.dat"
CPLFormFilename("/abc/xyz/","def.dat", NULL ) == "/abc/xyz/def.dat"

```

**Parameters**

<i>pszPath</i>	directory path to the directory containing the file. This may be relative or absolute, and may have a trailing path separator or not. May be NULL.
<i>pszBasename</i>	file basename. May optionally have path and/or extension. Must <i>NOT</i> be NULL.
<i>pszExtension</i>	file extension, optionally including the period. May be NULL.

**Returns**

a fully formed filename in an internal static string. Do not modify or free the returned string. The string may be destroyed by the next CPL call.

References CPLStrlcat(), and CPLStrlcpy().

Referenced by OGRSFDriverRegistrar::AutoLoadDrivers(), CPLCorrespondingPaths(), CPLFormCIFilename(), CPLGenerateTempFilename(), and CPLUnlinkTree().

### 13.1.2.15 `const char* CPLGenerateTempFilename ( const char * pszStem )`

Generate temporary file name.

Returns a filename that may be used for a temporary file. The location of the file tries to follow operating system semantics but may be forced via the CPL\_TMPDIR configuration option.

**Parameters**

<i>pszStem</i>	if non-NULL this will be part of the filename.
----------------	--

**Returns**

a filename which is valid till the next CPL call in this thread.

References CPLFormFilename(), and CPLGetConfigOption().

### 13.1.2.16 `const char* CPLGetBasename ( const char * pszFullFilename )`

Extract basename (non-directory, non-extension) portion of filename.

Returns a string containing the file basename portion of the passed name. If there is no basename (passed value ends in trailing directory separator, or filename starts with a dot) an empty string is returned.

```

CPLGetBasename( "abc/def.xyz" ) == "def"
CPLGetBasename( "abc/def" ) == "def"
CPLGetBasename( "abc/def/" ) == ""

```

**Parameters**

<i>pszFullFilename</i>	the full filename potentially including a path.
------------------------	---

**Returns**

just the non-directory, non-extension portion of the path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call.

References CPLStrlcpy().

Referenced by OGRSFDriverRegistrar::AutoLoadDrivers(), and CPLCorrespondingPaths().

### 13.1.2.17 `const char* CPLGetConfigOption ( const char * pszKey, const char * pszDefault )`

Get the value of a configuration option.

The value is the value of a (key, value) option set with **CPLSetConfigOption()** (p. ??). If the given option was not defined with **CPLSetConfigOption()** (p. ??), it tries to find it in environment variables.

Note: the string returned by **CPLGetConfigOption()** (p. ??) might be short-lived, and in particular it will become invalid after a call to **CPLSetConfigOption()** (p. ??) with the same key.

To override temporarily a potentially existing option with a new value, you can use the following snippet :

```
// backup old value
const char* pszOldValTmp = CPLGetConfigOption(pszKey, NULL);
char* pszOldVal = pszOldValTmp ? CPLStrdup(pszOldValTmp) : NULL;
// override with new value
CPLSetConfigOption(pszKey, pszNewVal);
// do something usefull
// restore old value
CPLSetConfigOption(pszKey, pszOldVal);
CPLFree(pszOldVal);
```

#### Parameters

<i>pszKey</i>	the key of the option to retrieve
<i>pszDefault</i>	a default value if the key does not match existing defined options (may be NULL)

#### Returns

the value associated to the key, or the default value if not found

#### See Also

**CPLSetConfigOption()** (p. ??), <http://trac.osgeo.org/gdal/wiki/ConfigOptions>

Referenced by OGRGeometryFactory::approximateArcAngles(), OGRSFDriverRegistrar::AutoLoadDrivers(), CPLDebug(), CPLGenerateTempFilename(), CPLHTTPFetch(), OGRSpatialReference::exportToProj4(), GOA2GetAccessToken(), GOA2GetAuthorizationURL(), GOA2GetRefreshToken(), OGRSpatialReference::morphFromESRI(), OGR\_G\_CreateFromGML(), OGRGeometryFactory::organizePolygons(), OGRSFDriverRegistrar::RegisterDriver(), OGRFeature::SetField(), and OGRLineString::transform().

### 13.1.2.18 `char* CPLGetCurrentDir ( void )`

Get the current working directory name.

#### Returns

a pointer to buffer, containing current working directory path or NULL in case of error. User is responsible to free that buffer after usage with CPLFree() function. If HAVE\_GETCWD macro is not defined, the function returns NULL.

References CPLMalloc().

### 13.1.2.19 `const char* CPLGetDirname ( const char * pszFilename )`

Extract directory path portion of filename.

Returns a string containing the directory path portion of the passed filename. If there is no path in the passed filename the dot will be returned. It is the only difference from **CPLGetPath()** (p. ??).

```

CPLGetDirname( "abc/def.xyz" ) == "abc"
CPLGetDirname( "/abc/def/" ) == "/abc/def"
CPLGetDirname( "/" ) == "/"
CPLGetDirname( "/abc/def" ) == "/abc"
CPLGetDirname( "abc" ) == "."

```

**Parameters**

<i>pszFilename</i>	the filename potentially including a path.
--------------------	--

**Returns**

Path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call. The returned will generally not contain a trailing path separator.

References CPLStrlcpy().

Referenced by OGRSFDriverRegistrar::AutoLoadDrivers().

**13.1.2.20 int CPLGetExecPath ( char \* *pszPathBuf*, int *nMaxLength* )**

Fetch path of executable.

The path to the executable currently running is returned. This path includes the name of the executable. Currently this only works on win32 and linux platforms. The returned path is UTF-8 encoded.

**Parameters**

<i>pszPathBuf</i>	the buffer into which the path is placed.
<i>nMaxLength</i>	the buffer size, MAX_PATH+1 is suggested.

**Returns**

FALSE on failure or TRUE on success.

Referenced by OGRSFDriverRegistrar::AutoLoadDrivers().

**13.1.2.21 const char\* CPLGetExtension ( const char \* *pszFullFilename* )**

Extract filename extension from full filename.

Returns a string containing the extension portion of the passed name. If there is no extension (the filename has no dot) an empty string is returned. The returned extension will not include the period.

```

CPLGetExtension( "abc/def.xyz" ) == "xyz"
CPLGetExtension( "abc/def" ) == ""

```

**Parameters**

<i>pszFullFilename</i>	the full filename potentially including a path.
------------------------	---

**Returns**

just the extension portion of the path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call.

References CPLStrlcpy().

Referenced by OGRSFDriverRegistrar::AutoLoadDrivers(), and OGRGeocodeCreateSession().

**13.1.2.22** `const char* CPLGetFilename ( const char * pszFullFilename )`

Extract non-directory portion of filename.

Returns a string containing the bare filename portion of the passed filename. If there is no filename (passed value ends in trailing directory separator) an empty string is returned.

```
CPLGetFilename( "abc/def.xyz" ) == "def.xyz"
CPLGetFilename( "/abc/def/" ) == ""
CPLGetFilename( "abc/def" ) == "def"
```

**Parameters**

<i>pszFullFilename</i>	the full filename potentially including a path.
------------------------	---

**Returns**

just the non-directory portion of the path (points back into original string).

Referenced by CPLCheckForFile(), and CPLCorrespondingPaths().

**13.1.2.23** `const char* CPLGetPath ( const char * pszFilename )`

Extract directory path portion of filename.

Returns a string containing the directory path portion of the passed filename. If there is no path in the passed filename an empty string will be returned (not NULL).

```
CPLGetPath( "abc/def.xyz" ) == "abc"
CPLGetPath( "/abc/def/" ) == "/abc/def"
CPLGetPath( "/" ) == "/"
CPLGetPath( "/abc/def" ) == "/abc"
CPLGetPath( "abc" ) == ""
```

**Parameters**

<i>pszFilename</i>	the filename potentially including a path.
--------------------	--

**Returns**

Path in an internal string which must not be freed. The string may be destroyed by the next CPL filename handling call. The returned will generally not contain a trailing path separator.

References CPLStrncpy().

Referenced by CPLCorrespondingPaths().

**13.1.2.24** `CPLSharedFileInfo* CPLGetSharedList ( int * pnCount )`

Fetch list of open shared files.

**Parameters**

<i>pnCount</i>	place to put the count of entries.
----------------	------------------------------------

**Returns**

the pointer to the first in the array of shared file info structures.

### 13.1.2.25 void\* CPLGetSymbol ( const char \* *pszLibrary*, const char \* *pszSymbolName* )

Fetch a function pointer from a shared library / DLL.

This function is meant to abstract access to shared libraries and DLLs and performs functions similar to `dlopen()/dlsym()` on Unix and `LoadLibrary()` / `GetProcAddress()` on Windows.

If no support for loading entry points from a shared library is available this function will always return NULL. Rules on when this function issues a **CPLError()** (p. ??) or not are not currently well defined, and will have to be resolved in the future.

Currently **CPLGetSymbol()** (p. ??) doesn't try to:

- prevent the reference count on the library from going up for every request, or given any opportunity to unload the library.
- Attempt to look for the library in non-standard locations.
- Attempt to try variations on the symbol name, like pre-pending or post-pending an underscore.

Some of these issues may be worked on in the future.

#### Parameters

<i>pszLibrary</i>	the name of the shared library or DLL containing the function. May contain path to file. If not system supplies search paths will be used.
<i>pszSymbolName</i>	the name of the function to fetch a pointer to.

#### Returns

A pointer to the function if found, or NULL if the function isn't found, or the shared library can't be loaded.

References **CPLError()**.

Referenced by `OGRSFDriverRegistrar::AutoLoadDrivers()`.

### 13.1.2.26 int CPLIsFilenameRelative ( const char \* *pszFilename* )

Is filename relative or absolute?

The test is filesystem convention agnostic. That is it will test for Unix style and windows style path conventions regardless of the actual system in use.

#### Parameters

<i>pszFilename</i>	the filename with path to test.
--------------------	---------------------------------

#### Returns

TRUE if the filename is relative or FALSE if it is absolute.

Referenced by `CPLExtractRelativePath()`, and `CPLProjectRelativeFilename()`.

### 13.1.2.27 void\* CPLMalloc ( size\_t *nSize* )

Safe version of `malloc()`.

This function is like the C library `malloc()`, but raises a `CE_Fatal` error with **CPLError()** (p. ??) if it fails to allocate the desired memory. It should be used for small memory allocations that are unlikely to fail and for which the application is unwilling to test for out of memory conditions. It uses `VSIMalloc()` to get the memory, so any hooking of `VSI-Malloc()` will apply to **CPLMalloc()** (p. ??) as well. `CPLFree()` or `VSIFree()` can be used free memory allocated by **CPLMalloc()** (p. ??).

## Parameters

<i>nSize</i>	size (in bytes) of memory block to allocate.
--------------	--

## Returns

pointer to newly allocated memory, only NULL if *nSize* is zero.

References `CPLEmergencyError()`, and `CPLError()`.

Referenced by `CPLStringList::AddNameValue()`, `OGRDataSource::CopyLayer()`, `CPLBinaryToHex()`, `CPLCalloc()`, `CPLEscapeString()`, `CPLForceToASCII()`, `CPLFormCIFilename()`, `CPLGetCurrentDir()`, `CPLHashSetNew()`, `CPLHexToBinary()`, `CPLListAppend()`, `CPLListInsert()`, `CPLParseNameValue()`, `CPLPrintTime()`, `CPLPushErrorHandlerEx()`, `CPLQuadTreeCreate()`, `CPLScanDouble()`, `CPLScanLong()`, `CPLScanString()`, `CPLScanUIntBig()`, `CPLScanULong()`, `CPLSerializeXMLTree()`, `CPLStrdup()`, `CPLUnescapeString()`, `CSLDuplicate()`, `CSLSetNameValue()`, `CSLSetNameValueSeparator()`, `OGRSpatialReference::exportToPCI()`, `OGRSpatialReference::exportToUSGS()`, `OGR_SRSNode::exportToWkt()`, `CPLODBCStatement::Fetch()`, `OGRSpatialReference::importFromC-RSURL()`, `OGRSpatialReference::importFromPanorama()`, `OGRSpatialReference::importFromPCI()`, `CPLODBCDriverInstaller::InstallDriver()`, `OGRSpatialReference::morphToESRI()`, `OGR_G_ExportToGMLEx()`, `OGR_G_ExportToKML()`, `OGRFeature::OGRFeature()`, `OGRLayer::ReorderField()`, `OGRFeatureDefn::ReorderFieldDefns()`, `OGRFeature::SetField()`, `OGRSpatialReference::SetFromUserInput()`, `CPLStringList::SetNameValue()`, and `OGR-Proj4CT::Transform()`.

### 13.1.2.28 FILE\* CPLOpenShared ( const char \* *pszFilename*, const char \* *pszAccess*, int *bLarge* )

Open a shared file handle.

Some operating systems have limits on the number of file handles that can be open at one time. This function attempts to maintain a registry of already open file handles, and reuse existing ones if the same file is requested by another part of the application.

Note that access is only shared for access types "r", "rb", "r+" and "rb+". All others will just result in direct `VSIOpen()` calls. Keep in mind that a file is only reused if the file name is exactly the same. Different names referring to the same file will result in different handles.

The `VSIOpen()` or **`VSIFOpenL()`** (p. ??) function is used to actually open the file, when an existing file handle can't be shared.

## Parameters

<i>pszFilename</i>	the name of the file to open.
<i>pszAccess</i>	the normal <code>fopen()</code> / <code>VSIOpen()</code> style access string.
<i>bLarge</i>	If TRUE <b><code>VSIFOpenL()</code></b> (p. ??) (for large files) will be used instead of <code>VSIOpen()</code> .

## Returns

a file handle or NULL if opening fails.

References `CPLRealloc()`, `CPLStrdup()`, and `VSIFOpenL()`.

### 13.1.2.29 double CPLPackedDMSToDec ( double *dfPacked* )

Convert a packed DMS value (DDDMMMSSS.SS) into decimal degrees.

This function converts a packed DMS angle to seconds. The standard packed DMS format is:

degrees \* 1000000 + minutes \* 1000 + seconds

Example: ang = 120025045.25 yields deg = 120 min = 25 sec = 45.25

The algorithm used for the conversion is as follows:

1. The absolute value of the angle is used.
2. The degrees are separated out:  $\text{deg} = \text{ang} / 1000000$  (fractional portion truncated)
3. The minutes are separated out:  $\text{min} = (\text{ang} - \text{deg} * 1000000) / 1000$  (fractional portion truncated)
4. The seconds are then computed:  $\text{sec} = \text{ang} - \text{deg} * 1000000 - \text{min} * 1000$
5. The total angle in seconds is computed:  $\text{sec} = \text{deg} * 3600.0 + \text{min} * 60.0 + \text{sec}$
6. The sign of sec is set to that of the input angle.

Packed DMS values used by the USGS GCTP package and probably by other software.

NOTE: This code does not validate input value. If you give the wrong value, you will get the wrong result.

#### Parameters

<i>dfPacked</i>	Angle in packed DMS format.
-----------------	-----------------------------

#### Returns

Angle in decimal degrees.

Referenced by OGRSpatialReference::importFromUSGS().

#### 13.1.2.30 int CPLPrintDouble ( char \* *pszBuffer*, const char \* *pszFormat*, double *dfValue*, const char \* *pszLocale* )

Print double value into specified string buffer. Exponential character flag 'E' (or 'e') will be replaced with 'D', as in Fortran. Resulting string will not to be NULL-terminated.

#### Parameters

<i>pszBuffer</i>	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
<i>pszFormat</i>	Format specifier (for example, "%16.9E").
<i>dfValue</i>	Numerical value to print.
<i>pszLocale</i>	Pointer to a character string containing locale name ("C", "POSIX", "us_US", "ru_RU.KOI8-R" etc.). If NULL we will not manipulate with locale settings and current process locale will be used for printing. With the pszLocale option we can control what exact locale will be used for printing a numeric value to the string (in most cases it should be C/POSIX).

#### Returns

Number of characters printed.

References CPLPrintString().

#### 13.1.2.31 int CPLPrintInt32 ( char \* *pszBuffer*, GInt32 *iValue*, int *nMaxLen* )

Print GInt32 value into specified string buffer. This string will not be NULL-terminated.

#### Parameters

<i>pszBuffer</i>	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
------------------	--

<i>iValue</i>	Numerical value to print.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than <i>nMaxLen</i> , it will be truncated.

**Returns**

Number of characters printed.

References CPLPrintString().

Referenced by OGRSpatialReference::exportToPCI().

### 13.1.2.32 int CPLPrintPointer ( char \* *pszBuffer*, void \* *pValue*, int *nMaxLen* )

Print pointer value into specified string buffer. This string will not be NULL-terminated.

**Parameters**

<i>pszBuffer</i>	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
<i>pValue</i>	Pointer to ASCII encode.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than <i>nMaxLen</i> , it will be truncated.

**Returns**

Number of characters printed.

References CPLPrintString().

### 13.1.2.33 int CPLPrintString ( char \* *pszDest*, const char \* *pszSrc*, int *nMaxLen* )

Copy the string pointed to by *pszSrc*, NOT including the terminating '\0' character, to the array pointed to by *pszDest*.

**Parameters**

<i>pszDest</i>	Pointer to the destination string buffer. Should be large enough to hold the resulting string.
<i>pszSrc</i>	Pointer to the source buffer.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than <i>nMaxLen</i> , it will be truncated.

**Returns**

Number of characters printed.

Referenced by CPLPrintDouble(), CPLPrintInt32(), CPLPrintPointer(), CPLPrintTime(), and CPLPrintUIntBig().

### 13.1.2.34 int CPLPrintStringFill ( char \* *pszDest*, const char \* *pszSrc*, int *nMaxLen* )

Copy the string pointed to by *pszSrc*, NOT including the terminating '\0' character, to the array pointed to by *pszDest*. Remainder of the destination string will be filled with space characters. This is only difference from the PrintString().

**Parameters**



<i>pszDest</i>	Pointer to the destination string buffer. Should be large enough to hold the resulting string.
<i>pszSrc</i>	Pointer to the source buffer.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

**Returns**

Number of characters printed.

Referenced by OGRSpatialReference::exportToPCI().

**13.1.2.35** int CPLPrintTime ( char \* *pszBuffer*, int *nMaxLen*, const char \* *pszFormat*, const struct tm \* *poBrokenTime*, const char \* *pszLocale* )

Print specified time value accordingly to the format options and specified locale name. This function does following:

- if locale parameter is not NULL, the current locale setting will be stored and replaced with the specified one;
- format time value with the strftime(3) function;
- restore back current locale, if was saved.

**Parameters**

<i>pszBuffer</i>	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.
<i>pszFormat</i>	Controls the output format. Options are the same as for strftime(3) function.
<i>poBrokenTime</i>	Pointer to the broken-down time structure. May be requested with the VSIGMTime() and VSILocalTime() functions.
<i>pszLocale</i>	Pointer to a character string containing locale name ("C", "POSIX", "us_US", "ru_RU.KOI8-R" etc.). If NULL we will not manipulate with locale settings and current process locale will be used for printing. Be aware that it may be unsuitable to use current locale for printing time, because all names will be printed in your native language, as well as time format settings also may be adjusted differently from the C/POSIX defaults. To solve these problems this option was introduced.

**Returns**

Number of characters printed.

References CPLMalloc(), and CPLPrintString().

**13.1.2.36** int CPLPrintUIntBig ( char \* *pszBuffer*, GUIntBig *iValue*, int *nMaxLen* )

Print GUIntBig value into specified string buffer. This string will not be NULL-terminated.

**Parameters**

<i>pszBuffer</i>	Pointer to the destination string buffer. Should be large enough to hold the resulting string. Note, that the string will not be NULL-terminated, so user should do this himself, if needed.
------------------	--

<i>iValue</i>	Numerical value to print.
<i>nMaxLen</i>	Maximum length of the resulting string. If string length is greater than nMaxLen, it will be truncated.

#### Returns

Number of characters printed.

References CPLPrintString().

#### 13.1.2.37 `const char* CPLProjectRelativeFilename ( const char * pszProjectDir, const char * pszSecondaryFilename )`

Find a file relative to a project file.

Given the path to a "project" directory, and a path to a secondary file referenced from that project, build a path to the secondary file that the current application can use. If the secondary path is already absolute, rather than relative, then it will be returned unaltered.

Examples:

```
CPLProjectRelativeFilename("abc/def", "tmp/abc.gif") == "abc/def/tmp/abc.gif"
CPLProjectRelativeFilename("abc/def", "/tmp/abc.gif") == "/tmp/abc.gif"
CPLProjectRelativeFilename("/xy", "abc.gif") == "/xy/abc.gif"
CPLProjectRelativeFilename("/abc/def", "../abc.gif") == "/abc/def/../abc.gif"
CPLProjectRelativeFilename("C:\\WIN", "abc.gif") == "C:\\WIN\\abc.gif"
```

#### Parameters

<i>pszProjectDir</i>	the directory relative to which the secondary files path should be interpreted.
<i>pszSecondaryFilename</i>	the filename (potentially with path) that is to be interpreted relative to the project directory.

#### Returns

a composed path to the secondary file. The returned string is internal and should not be altered, freed, or depending on past the next CPL call.

References CPLIsFilenameRelative(), CPLStrlcat(), and CPLStrlcpy().

#### 13.1.2.38 `const char* CPLReadLine ( FILE * fp )`

Simplified line reading from text file.

Read a line of text from the given file handle, taking care to capture CR and/or LF and strip off ... equivalent of DKReadLine(). Pointer to an internal buffer is returned. The application shouldn't free it, or depend on it's value past the next call to **CPLReadLine()** (p. ??).

Note that **CPLReadLine()** (p. ??) uses VSIFGets(), so any hooking of VSI file services should apply to **CPLReadLine()** (p. ??) as well.

**CPLReadLine()** (p. ??) maintains an internal buffer, which will appear as a single block memory leak in some circumstances. **CPLReadLine()** (p. ??) may be called with a NULL FILE \* at any time to free this working buffer.

#### Parameters

<i>fp</i>	file pointer opened with VSIFOpen().
-----------	--------------------------------------

**Returns**

pointer to an internal buffer containing a line of text read from the file or NULL if the end of file was encountered.

References CPLFGets().

Referenced by OGRSpatialReference::importFromDict().

### 13.1.2.39 `const char* CPLReadLine2L ( VSILFILE * fp, int nMaxCars, char ** papszOptions )`

Simplified line reading from text file.

Similar to **CPLReadLine()** (p. ??), but reading from a large file API handle.

**Parameters**

<i>fp</i>	file pointer opened with <b>VSIFOpenL()</b> (p. ??).
<i>nMaxCars</i>	maximum number of characters allowed, or -1 for no limit.
<i>papszOptions</i>	NULL-terminated array of options. Unused for now.

**Returns**

pointer to an internal buffer containing a line of text read from the file or NULL if the end of file was encountered or the maximum number of characters allowed reached.

**Since**

GDAL 1.7.0

References CPLError(), VSIFReadL(), VSIFSeekL(), and VSIFTellL().

Referenced by CPLReadLineL(), and CSLLoad2().

### 13.1.2.40 `const char* CPLReadLineL ( VSILFILE * fp )`

Simplified line reading from text file.

Similar to **CPLReadLine()** (p. ??), but reading from a large file API handle.

**Parameters**

<i>fp</i>	file pointer opened with <b>VSIFOpenL()</b> (p. ??).
-----------	--

**Returns**

pointer to an internal buffer containing a line of text read from the file or NULL if the end of file was encountered.

References CPLReadLine2L().

Referenced by CSLLoad2().

### 13.1.2.41 `void* CPLRealloc ( void * pData, size_t nNewSize )`

Safe version of realloc().

This function is like the C library realloc(), but raises a CE\_Fatal error with **CPLError()** (p. ??) if it fails to allocate the desired memory. It should be used for small memory allocations that are unlikely to fail and for which the application

is unwilling to test for out of memory conditions. It uses `VSIRealloc()` to get the memory, so any hooking of `VSIRealloc()` will apply to **CPLRealloc()** (p. ??) as well. `CPLFree()` or `VSIFree()` can be used free memory allocated by **CPLRealloc()** (p. ??).

It is also safe to pass `NULL` in as the existing memory block for **CPLRealloc()** (p. ??), in which case it uses `VSIMalloc()` to allocate a new block.

## Parameters

<i>pData</i>	existing memory block which should be copied to the new block.
<i>nNewSize</i>	new size (in bytes) of memory block to allocate.

## Returns

pointer to allocated memory, only NULL if nNewSize is zero.

References CPLEmergencyError(), and CPLError().

Referenced by OGRFeatureDefn::AddFieldDefn(), CPLHTTPParseMultipartMime(), CPLOpenShared(), CSL-TokenizeString2(), OGRDataSource::ExecuteSQL(), OGRMultiPoint::exportToWkt(), CPLDBCStatement::Fetch(), OGRSpatialReference::importFromESRI(), OGRPolygon::importFromWkt(), OGR\_SRSNode::InsertChild(), OGR-SFDriverRegistrar::RegisterDriver(), and OGRProj4CT::TransformEx().

#### 13.1.2.42 `const char* CPLResetExtension ( const char * pszPath, const char * pszExt )`

Replace the extension with the provided one.

## Parameters

<i>pszPath</i>	the input path, this string is not altered.
<i>pszExt</i>	the new extension to apply to the given path.

## Returns

an altered filename with the new extension. Do not modify or free the returned string. The string may be destroyed by the next CPL call.

References CPLStrcat(), and CPLStrncpy().

#### 13.1.2.43 `double CPLScanDouble ( const char * pszString, int nMaxLength )`

Extract double from string.

Scan up to a maximum number of characters from a string and convert the result to a double. This function uses **CPLAtof()** (p. ??) to convert string to double value, so it uses a comma as a decimal delimiter.

## Parameters

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

## Returns

Double value, converted from its ASCII form.

References CPLAtof(), and CPLMalloc().

#### 13.1.2.44 `long CPLScanLong ( const char * pszString, int nMaxLength )`

Scan up to a maximum number of characters from a string and convert the result to a long.

## Parameters

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

## Returns

Long value, converted from its ASCII form.

References CPLMalloc().

Referenced by OGRSpatialReference::importFromPCI(), and OGRParseDate().

#### 13.1.2.45 void\* CPLScanPointer ( const char \* *pszString*, int *nMaxLength* )

Extract pointer from string.

Scan up to a maximum number of characters from a string and convert the result to a pointer.

## Parameters

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

## Returns

pointer value, converted from its ASCII form.

References CPLScanUIntBig(), and CPLScanULong().

#### 13.1.2.46 char\* CPLScanString ( const char \* *pszString*, int *nMaxLength*, int *bTrimSpaces*, int *bNormalize* )

Scan up to a maximum number of characters from a given string, allocate a buffer for a new string and fill it with scanned characters.

## Parameters

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	The maximum number of character to read. Less characters will be read if a null character is encountered.
<i>bTrimSpaces</i>	If TRUE, trim ending spaces from the input string. Character considered as empty using isspace(3) function.
<i>bNormalize</i>	If TRUE, replace ':' symbol with the '_'. It is needed if resulting string will be used in CPL dictionaries.

## Returns

Pointer to the resulting string buffer. Caller responsible to free this buffer with CPLFree().

References CPLMalloc(), and CPLStrdup().

#### 13.1.2.47 GUIntBig CPLScanUIntBig ( const char \* *pszString*, int *nMaxLength* )

Extract big integer from string.

Scan up to a maximum number of characters from a string and convert the result to a GUIntBig.

## Parameters

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

## Returns

GUIntBig value, converted from its ASCII form.

References CPLMalloc().

Referenced by CPLScanPointer().

#### 13.1.2.48 unsigned long CPLScanULong ( const char \* *pszString*, int *nMaxLength* )

Scan up to a maximum number of characters from a string and convert the result to a unsigned long.

## Parameters

<i>pszString</i>	String containing characters to be scanned. It may be terminated with a null character.
<i>nMaxLength</i>	The maximum number of character to consider as part of the number. Less characters will be considered if a null character is encountered.

## Returns

Unsigned long value, converted from its ASCII form.

References CPLMalloc().

Referenced by CPLScanPointer().

#### 13.1.2.49 void CPLSetConfigOption ( const char \* *pszKey*, const char \* *pszValue* )

Set a configuration option for GDAL/OGR use.

Those options are defined as a (key, value) couple. The value corresponding to a key can be got later with the **CPLGetConfigOption()** (p. ??) method.

This mechanism is similar to environment variables, but options set with **CPLSetConfigOption()** (p. ??) overrides, for **CPLGetConfigOption()** (p. ??) point of view, values defined in the environment.

If **CPLSetConfigOption()** (p. ??) is called several times with the same key, the value provided during the last call will be used.

Options can also be passed on the command line of most GDAL utilities with the with '-config KEY VALUE'. For example, ogrinfo -config CPL\_DEBUG ON ~/data/test/point.shp

This function can also be used to clear a setting by passing NULL as the value (note: passing NULL will not unset an existing environment variable; it will just unset a value previously set by **CPLSetConfigOption()** (p. ??)).

## Parameters

<i>pszKey</i>	the key of the option
<i>pszValue</i>	the value of the option, or NULL to clear a setting.

## See Also

<http://trac.osgeo.org/gdal/wiki/ConfigOptions>

References CSLSetNameValue().

Referenced by CPLHTTPFetch().

### 13.1.2.50 void CPLSetThreadLocalConfigOption ( const char \* *pszKey*, const char \* *pszValue* )

Set a configuration option for GDAL/OGR use.

Those options are defined as a (key, value) couple. The value corresponding to a key can be got later with the **CPLGetConfigOption()** (p. ??) method.

This function sets the configuration option that only applies in the current thread, as opposed to **CPLSetConfigOption()** (p. ??) which sets an option that applies on all threads.

This function can also be used to clear a setting by passing NULL as the value (note: passing NULL will not unset an existing environment variable; it will just unset a value previously set by **CPLSetThreadLocalConfigOption()** (p. ??)).

#### Parameters

<i>pszKey</i>	the key of the option
<i>pszValue</i>	the value of the option, or NULL to clear a setting.

References CSLDestroy(), and CSLSetNameValue().

### 13.1.2.51 char\* CPLStrdup ( const char \* *pszString* )

Safe version of strdup() function.

This function is similar to the C library strdup() function, but if the memory allocation fails it will issue a CE\_Fatal error with **CPLError()** (p. ??) instead of returning NULL. It uses VSIStrdup(), so any hooking of that function will apply to **CPLStrdup()** (p. ??) as well. Memory allocated with **CPLStrdup()** (p. ??) can be freed with CPLFree() or VSIFree().

It is also safe to pass a NULL string into **CPLStrdup()** (p. ??). **CPLStrdup()** (p. ??) will allocate and return a zero length string (as opposed to a NULL string).

#### Parameters

<i>pszString</i>	input string to be duplicated. May be NULL.
------------------	---

#### Returns

pointer to a newly allocated copy of the string. Free with CPLFree() or VSIFree().

References CPLError(), and CPLMalloc().

Referenced by OGRStyleMgr::AddPart(), CPLStringList::AddString(), CPLCreateXMLNode(), CPLEscapeString(), CPLHTTPFetch(), CPLOpenShared(), CPLRecode(), CPLScanString(), CPLSetXMLValue(), CPLUnlinkTree(), CSLDuplicate(), CSLLoad2(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToPrettyWkt(), OGRSpatialReference::exportToProj4(), OGRSpatialReference::exportToWkt(), OGRPoint::exportToWkt(), OGRLineString::exportToWkt(), OGRPolygon::exportToWkt(), OGRGeometryCollection::exportToWkt(), OGRMultiPolygon::exportToWkt(), OGRMultiPoint::exportToWkt(), OGRMultiLineString::exportToWkt(), CPLODBCStatement::GetColumns(), OGRFeature::GetFieldAsString(), GOA2GetAccessToken(), GOA2GetAuthorizationURL(), GOA2GetRefreshToken(), OGRSpatialReference::importFromCRSURL(), OGRSpatialReference::importFromESRI(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::importFromURN(), OGRStyleMgr::InitStyleString(), CPLStringList::InsertString(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGR\_G\_ExportToGMLEx(), OGR\_G\_ExportToJsonEx(), OGR\_G\_ExportToKML(), OGR\_SRSNode::OGR\_SRSNode(), OGRFeatureDefn::OGRFeatureDefn(), OGRGeocodeCreateSession(), OGRUnionLayer::SetAttributeFilter(), OGRFeature::SetField(), OGRSpatialReference::SetFromUserInput(), OGRSpatialReference::SetLinearUnitsAndUpdateParameters(), OGRFieldDefn::SetName(), OGRFeature::SetStyleString(), OGR\_SRSNode::SetValue(), and VSIRReadDirRecursive().

### 13.1.2.52 char\* CPLStrlwr ( char \* *pszString* )

Convert each characters of the string to lower case.



For example, "ABcdE" will be converted to "abcde". This function is locale dependent.

## Parameters

<i>pszString</i>	input string to be converted.
------------------	-------------------------------

## Returns

pointer to the same string, pszString.

### 13.1.2.53 double CPLStrtod ( const char \* *nptr*, char \*\* *endptr* )

Converts ASCII string to floating point number.

This function converts the initial portion of the string pointed to by *nptr* to double floating point representation. This function does the same as standard `strtod(3)`, but does not take locale in account. That means, the decimal delimiter is always '.' (decimal point). Use **CPLStrtodDelim()** (p. ??) function if you want to specify custom delimiter. Also see notes for **CPLAtof()** (p. ??) function.

## Parameters

<i>nptr</i>	Pointer to string to convert.
<i>endptr</i>	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by <i>endptr</i> .

## Returns

Converted value, if any.

References CPLStrtodDelim().

Referenced by CPLAtof(), and OGRFeature::SetField().

### 13.1.2.54 double CPLStrtodDelim ( const char \* *nptr*, char \*\* *endptr*, char *point* )

Converts ASCII string to floating point number using specified delimiter.

This function converts the initial portion of the string pointed to by *nptr* to double floating point representation. This function does the same as standard `strtod(3)`, but does not take locale in account. Instead of locale defined decimal delimiter you can specify your own one. Also see notes for **CPLAtof()** (p. ??) function.

## Parameters

<i>nptr</i>	Pointer to string to convert.
<i>endptr</i>	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by <i>endptr</i> .
<i>point</i>	Decimal delimiter.

## Returns

Converted value, if any.

Referenced by CPLAtofDelim(), CPLAtofM(), CPLStrtod(), and CPLStrtofDelim().

### 13.1.2.55 float CPLStrtof ( const char \* *nptr*, char \*\* *endptr* )

Converts ASCII string to floating point number.

This function converts the initial portion of the string pointed to by *nptr* to single floating point representation. This function does the same as standard `strtof(3)`, but does not take locale in account. That means, the decimal delimiter is always '.' (decimal point). Use **CPLStrtofDelim()** (p. ??) function if you want to specify custom delimiter. Also see notes for **CPLAtof()** (p. ??) function.

## Parameters

<i>nptr</i>	Pointer to string to convert.
<i>endptr</i>	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by endptr.

## Returns

Converted value, if any.

References CPLStrtofDelim().

#### 13.1.2.56 float CPLStrtofDelim ( const char \* *nptr*, char \*\* *endptr*, char *point* )

Converts ASCII string to floating point number using specified delimiter.

This function converts the initial portion of the string pointed to by *nptr* to single floating point representation. This function does the same as standard strtod(3), but does not take locale in account. Instead of locale defined decimal delimiter you can specify your own one. Also see notes for **CPLAtof()** (p. ??) function.

## Parameters

<i>nptr</i>	Pointer to string to convert.
<i>endptr</i>	If is not NULL, a pointer to the character after the last character used in the conversion is stored in the location referenced by endptr.
<i>point</i>	Decimal delimiter.

## Returns

Converted value, if any.

References CPLStrtodDelim().

Referenced by CPLStrtof().

#### 13.1.2.57 int CPLUnlinkTree ( const char \* *pszPath* )

## Returns

0 on successful completion, -1 if function fails.

References CPLError(), CPLFormFilename(), CPLStrdup(), CSLDestroy(), VSIRmdir(), and VSIUnlink().

#### 13.1.2.58 void\* CPLZLibDeflate ( const void \* *ptr*, size\_t *nBytes*, int *nLevel*, void \* *outptr*, size\_t *nOutAvailableBytes*, size\_t \* *pnOutBytes* )

Compress a buffer with ZLib DEFLATE compression.

## Parameters

<i>ptr</i>	input buffer.
<i>nBytes</i>	size of input buffer in bytes.
<i>nLevel</i>	ZLib compression level (-1 for default).
<i>outptr</i>	output buffer, or NULL to let the function allocate it.

<i>nOutAvailable-Bytes</i>	size of output buffer if provided, or ignored.
<i>pnOutBytes</i>	pointer to a <code>size_t</code> , where to store the size of the output buffer.

**Returns**

the output buffer (to be freed with `VSIFree()` if not provided) or `NULL` in case of error.

**Since**

GDAL 1.10.0

13.1.2.59 `void* CPLZLibInflate ( const void * ptr, size_t nBytes, void * outptr, size_t nOutAvailableBytes, size_t * pnOutBytes )`

Uncompress a buffer compressed with ZLib DEFLATE compression.

**Parameters**

<i>ptr</i>	input buffer.
<i>nBytes</i>	size of input buffer in bytes.
<i>outptr</i>	output buffer, or <code>NULL</code> to let the function allocate it.
<i>nOutAvailable-Bytes</i>	size of output buffer if provided, or ignored.
<i>pnOutBytes</i>	pointer to a <code>size_t</code> , where to store the size of the output buffer.

**Returns**

the output buffer (to be freed with `VSIFree()` if not provided) or `NULL` in case of error.

**Since**

GDAL 1.10.0

## 13.2 cpl\_error.h File Reference

```
#include "cpl_port.h"
```

**Functions**

- void **CPLError** (CPLerr eErrClass, int err\_no, const char \*fmt,...)
- void **CPLEmergencyError** (const char \*)
- void **CPLErrorReset** (void)
- int **CPLGetLastErrorNo** (void)
- CPLerr **CPLGetLastErrorType** (void)
- const char \* **CPLGetLastErrorMsg** (void)
- void \* **CPLGetErrorHandlerUserData** (void)
- CPLErrorHandler **CPLSetErrorHandler** (CPLErrorHandler)
- CPLErrorHandler **CPLSetErrorHandlerEx** (CPLErrorHandler, void \*)
- void **CPLPushErrorHandler** (CPLErrorHandler)
- void **CPLPushErrorHandlerEx** (CPLErrorHandler, void \*)
- void **CPLPopErrorHandler** (void)
- void **CPLDebug** (const char \*, const char \*,...)
- void **\_CPLAssert** (const char \*, const char \*, int)

### 13.2.1 Detailed Description

CPL error handling services.

### 13.2.2 Function Documentation

#### 13.2.2.1 void \_CPLAssert ( const char \* *pszExpression*, const char \* *pszFile*, int *iLine* )

Report failure of a logical assertion.

Applications would normally use the CPLAssert() macro which expands into code calling **\_CPLAssert()** (p. ??) only if the condition fails. **\_CPLAssert()** (p. ??) will generate a CE\_Fatal error call to **CPL\_Error()** (p. ??), indicating the file name, and line number of the failed assertion, as well as containing the assertion itself.

There is no reason for application code to call **\_CPLAssert()** (p. ??) directly.

References CPL\_Error().

#### 13.2.2.2 void CPLDebug ( const char \* *pszCategory*, const char \* *pszFormat*, ... )

Display a debugging message.

The category argument is used in conjunction with the CPL\_DEBUG environment variable to establish if the message should be displayed. If the CPL\_DEBUG environment variable is not set, no debug messages are emitted (use CPL\_Error(CE\_Warning,...) to ensure messages are displayed). If CPL\_DEBUG is set, but is an empty string or the word "ON" then all debug messages are shown. Otherwise only messages whose category appears somewhere within the CPL\_DEBUG value are displayed (as determined by strstr()).

Categories are usually an identifier for the subsystem producing the error. For instance "GDAL" might be used for the GDAL core, and "TIFF" for messages from the TIFF translator.

##### Parameters

<i>pszCategory</i>	name of the debugging message category.
<i>pszFormat</i>	printf() style format string for message to display. Remaining arguments are assumed to be for format.

References CPL\_GetConfigOption().

Referenced by OGRSFDriverRegistrar::AutoLoadDrivers(), CPLDumpSharedList(), CPLFGets(), CPLHTTPFetch(), CPLQuadTreeGetAdvisedMaxDepth(), CPLSetErrorHandlerEx(), CPLUnescapeString(), OGRGeometryFactory::createFromWkb(), OGRSpatialReference::Dereference(), OGRGeometry::Distance(), CPLODBCSession::EstablishSession(), OGRSpatialReference::exportToPanorama(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::exportToUSGS(), OGRLineString::exportToWkt(), OGRPolygon::exportToWkt(), OGRMultiPolygon::exportToWkt(), OGRMultiPoint::exportToWkt(), OGRMultiLineString::exportToWkt(), OGR\_SRSNode::FixupOrdering(), OGRSpatialReference::GetAxis(), CPLODBCStatement::GetTables(), GOA2GetAccessToken(), GOA2GetRefreshToken(), OGRSpatialReference::importFromESRI(), OGRSpatialReference::importFromOzi(), OGRSpatialReference::importFromPanorama(), OGRSpatialReference::importFromPCI(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::importFromUSGS(), CPLODBCDriverInstaller::InstallDriver(), OGRSpatialReference::morphFromESRI(), OGRSpatialReference::morphToESRI(), OGR\_Dr\_CreateDataSource(), OGRBuildPolygonFromEdges(), OGRSFDriverRegistrar::Open(), OGRGeometryFactory::organizePolygons(), OGRSpatialReference::SetFromUserInput(), OGRSpatialReference::SetGeocCS(), OGRSpatialReference::SetLocalCS(), OGRSpatialReference::SetProjCS(), OGRPolygon::transform(), OGRGeometryCollection::transform(), OGRSpatialReference::Validate(), and VSIGetMemFileBuffer().

#### 13.2.2.3 void CPLEmergencyError ( const char \* *pszMessage* )

Fatal error when things are bad.

This function should be called in an emergency situation where it is unlikely that a regular error report would work.

This would include in the case of heap exhaustion for even small allocations, or any failure in the process of reporting an error (such as TLS allocations).

This function should never return. After the error message has been reported as best possible, the application will abort() similarly to how **CPLError()** (p. ??) aborts on CE\_Fatal class errors.

#### Parameters

<i>pszMessage</i>	the error message to report.
-------------------	------------------------------

Referenced by CPLMalloc(), and CPLRealloc().

#### 13.2.2.4 void CPLError ( CPLErr eErrClass, int err\_no, const char \* fmt, ... )

Report an error.

This function reports an error in a manner that can be hooked and reported appropriate by different applications.

The effect of this function can be altered by applications by installing a custom error handling using **CPLSetErrorHandler()** (p. ??).

The eErrClass argument can have the value CE\_Warning indicating that the message is an informational warning, CE\_Failure indicating that the action failed, but that normal recover mechanisms will be used or CE\_Fatal meaning that a fatal error has occurred, and that **CPLError()** (p. ??) should not return.

The default behaviour of **CPLError()** (p. ??) is to report errors to stderr, and to abort() after reporting a CE\_Fatal error. It is expected that some applications will want to suppress error reporting, and will want to install a C++ exception, or longjmp() approach to no local fatal error recovery.

Regardless of how application error handlers or the default error handler choose to handle an error, the error number, and message will be stored for recovery with **CPLGetLastErrorNo()** (p. ??) and **CPLGetLastErrorMsg()** (p. ??).

#### Parameters

<i>eErrClass</i>	one of CE_Warning, CE_Failure or CE_Fatal.
<i>err_no</i>	the error number (CPL_*_) from <b>cpl_error.h</b> (p. ??).
<i>fmt</i>	a printf() style format string. Any additional arguments will be treated as arguments to fill in this format in a manner similar to printf().

Referenced by \_CPLAssert(), OGRLayer::AlterFieldDefn(), OGRGeometry::Boundary(), OGRGeometry::Buffer(), OGRGeometry::Centroid(), OGRLayer::Clip(), OGRGeometry::Contains(), OGRGeometry::ConvexHull(), OGRSFDriver::CopyDataSource(), OGRDataSource::CopyLayer(), CPLCloseShared(), CPLCorrespondingPaths(), CPLEscapeString(), CPLGetSymbol(), CPLHTTPFetch(), CPLHTTPParseMultipartMime(), CPLMalloc(), CPLParseXMLFile(), CPLParseXMLString(), CPLReadLine2L(), CPLRealloc(), CPLSerializeXMLTreeToFile(), CPLStrdup(), CPLUnlinkTree(), OGRSFDriver::CreateDataSource(), OGRUnionLayer::CreateFeature(), OGRLayer::CreateField(), OGRDataSource::CreateLayer(), OGRGeometry::Crosses(), CSLLoad2(), OGRSFDriver::DeleteDataSource(), OGRLayer::DeleteField(), OGRDataSource::DeleteLayer(), OGRGeometry::Difference(), OGRGeometry::Disjoint(), OGRGeometry::Distance(), OGRLayer::Erase(), OGRDataSource::ExecuteSQL(), OGRGeometry::exportToJson(), OGRGeometry::exportToKML(), OGRSpatialReference::exportToMICoordSys(), OGRSpatialReference::exportToProj4(), CPLDBCStatement::Fetch(), OGRFeatureDefn::GetFieldDefn(), GOA2GetAccessToken(), GOA2GetRefreshToken(), OGRLayer::Identity(), OGRSpatialReference::importFromCRSURL(), OGRSpatialReference::importFromEPSGA(), OGRSpatialReference::importFromMICoordSys(), OGRSpatialReference::importFromOzi(), OGRSpatialReference::importFromPanorama(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::importFromUrl(), OGRSpatialReference::importFromURN(), OGRSpatialReference::importFromUSGS(), OGRLineString::importFromWkb(), OGRPolygon::importFromWkb(), OGRSpatialReference::importFromWMSAUTO(), CPLStringList::InsertStringDirectly(), OGRLayer::Intersection(), OGRGeometry::Intersection(), OGR\_DS\_CreateLayer(), OGR\_F\_IsFieldSet(), OGR\_G\_AddPoint(), OGR\_G\_AddPoint\_2D(), OGR\_G\_Area(), OGR\_G\_Centroid(), OGR\_G\_CreateFromGML(), OGR\_G\_Equals(), OGR\_G\_GetGeometryRef(), OGR\_G\_GetPoint(), OGR\_G\_GetPoints(), OGR\_G\_GetX(), OGR\_G\_GetY(), OGR\_G\_GetZ(), OGR\_G\_Length(), OGR\_G\_PointOnSurface(), OGR\_G\_RemoveGeometry(), OGR\_G\_Segmentize(), OGR\_G\_SetPoint(), OGR\_G\_SetPoint\_2D(), OGRBuildPolygonFromEdges(), OGRCreateCoordinateTransformation(), OGRGeocode(), OGRGeocodeCreateSession(), OGRGeocodeReverse(), OGRParseDate(), OGRGeometryFactory::organizePolygons(), OG-

RGeometry::Overlaps(), OGRGeometry::Polygonize(), OGRLayer::ReorderField(), OGRLayer::ReorderFields(), OGRLineString::segmentize(), OGRSpatialReference::SetCompoundCS(), OGRSpatialReference::SetEckert(), OGRUnionLayer::SetFeature(), OGRFeature::SetField(), OGRLineString::setNumPoints(), OGRSpatialReference::SetStatePlane(), OGRSpatialReference::SetWagner(), OGRGeometry::Simplify(), OGRGeometry::Simplify-PreserveTopology(), OGRLayer::SymDifference(), OGRGeometry::SymDifference(), OGRGeometry::Touches(), OGRLineString::transform(), OGRProj4CT::TransformEx(), OGRLayer::Union(), OGRGeometry::Union(), OGRGeometry::UnionCascaded(), OGRLayer::Update(), VSIMalloc2(), VSIMalloc3(), and OGRGeometry::Within().

#### 13.2.2.5 void CPLErrorReset ( void )

Erase any traces of previous errors.

This is normally used to ensure that an error which has been recovered from does not appear to be still in play with high level functions.

Referenced by OGRDataSource::CopyLayer(), CPLParseXMLString(), CSLLoad2(), OGRSpatialReference::importFromUrl(), and OGRSFDriverRegistrar::Open().

#### 13.2.2.6 void\* CPLGetErrorHandlerUserData ( void )

Fetch the user data for the error context

Fetches the user data for the current error context. You can set the user data for the error context when you add your handler by issuing **CPLSetErrorHandlerEx()** (p. ??) and **CPLPushErrorHandlerEx()** (p. ??). Note that user data is primarily intended for providing context within error handlers themselves, but they could potentially be abused in other useful ways with the usual caveat emptor understanding.

##### Returns

the user data pointer for the error context

#### 13.2.2.7 const char\* CPLGetLastErrorMsg ( void )

Get the last error message.

Fetches the last error message posted with **CPLError()** (p. ??), that hasn't been cleared by **CPLErrorReset()** (p. ??). The returned pointer is to an internal string that should not be altered or freed.

##### Returns

the last error message, or NULL if there is no posted error message.

#### 13.2.2.8 int CPLGetLastErrorNo ( void )

Fetch the last error number.

Fetches the last error number posted with **CPLError()** (p. ??), that hasn't been cleared by **CPLErrorReset()** (p. ??). This is the error number, not the error class.

##### Returns

the error number of the last error to occur, or CPLE\_None (0) if there are no posted errors.

Referenced by OGRSpatialReference::importFromUrl().

### 13.2.2.9 CPLerr CPLGetLastErrorType ( void )

Fetch the last error type.

Fetches the last error type posted with **CPL\_Error()** (p. ??), that hasn't been cleared by **CPL\_ErrorReset()** (p. ??). This is the error class, not the error number.

#### Returns

the error type of the last error to occur, or CE\_None (0) if there are no posted errors.

Referenced by CPLParseXMLString(), and OGRSFDriverRegistrar::Open().

### 13.2.2.10 void CPLPopErrorHandler ( void )

Pop error handler off stack.

Discards the current error handler on the error handler stack, and restores the one in use before the last **CPLPushErrorHandler()** (p. ??) call. This method has no effect if there are no error handlers on the current threads error handler stack.

### 13.2.2.11 void CPLPushErrorHandler ( CPLErrorHandler pfnErrorHandlerNew )

Push a new CPL\_Error handler.

This pushes a new error handler on the thread-local error handler stack. This handler will be used until removed with **CPLPopErrorHandler()** (p. ??).

The **CPL\_SetErrorHandler()** (p. ??) docs have further information on how CPL\_Error handlers work.

#### Parameters

<i>pfnErrorHandler-New</i>	new error handler function.
----------------------------	-----------------------------

References CPLPushErrorHandlerEx().

### 13.2.2.12 void CPLPushErrorHandlerEx ( CPLErrorHandler pfnErrorHandlerNew, void \* pUserData )

Push a new CPL\_Error handler with user data on the error context.

This pushes a new error handler on the thread-local error handler stack. This handler will be used until removed with **CPLPopErrorHandler()** (p. ??). Obtain the user data back by using CPLGetErrorContext().

The **CPL\_SetErrorHandler()** (p. ??) docs have further information on how CPL\_Error handlers work.

#### Parameters

<i>pfnErrorHandler-New</i>	new error handler function.
<i>pUserData</i>	User data to put on the error context.

References CPLMalloc().

Referenced by CPLPushErrorHandler().

### 13.2.2.13 CPLErrorHandler CPLSetErrorHandler ( CPLErrorHandler pfnErrorHandlerNew )

Install custom error handler.

Allow the library's user to specify his own error handler function. A valid error handler is a C function with the following prototype:



```
void MyErrorHandler(CPLErr eErrClass, int err_no, const char *msg)
```

Pass NULL to come back to the default behavior. The default behaviour (CPLDefaultErrorHandler()) is to write the message to stderr.

The msg will be a partially formatted error message not containing the "ERROR %d:" portion emitted by the default handler. Message formatting is handled by **CPL\_Error()** (p. ??) before calling the handler. If the error handler function is passed a CE\_Fatal class error and returns, then **CPL\_Error()** (p. ??) will call abort(). Applications wanting to interrupt this fatal behaviour will have to use longjmp(), or a C++ exception to indirectly exit the function.

Another standard error handler is CPLQuietErrorHandler() which doesn't make any attempt to report the passed error or warning messages but will process debug messages via CPLDefaultErrorHandler.

Note that error handlers set with **CPL\_SetErrorHandler()** (p. ??) apply to all threads in an application, while error handlers set with CPLPushErrorHandler are thread-local. However, any error handlers pushed with CPLPushErrorHandler (and not removed with CPLPopErrorHandler) take precedence over the global error handlers set with **CPL\_SetErrorHandler()** (p. ??). Generally speaking **CPL\_SetErrorHandler()** (p. ??) would be used to set a desired global error handler, while **CPLPushErrorHandler()** (p. ??) would be used to install a temporary local error handler, such as CPLQuietErrorHandler() to suppress error reporting in a limited segment of code.

#### Parameters

<i>pfnErrorHandler-New</i>	new error handler function.
----------------------------	-----------------------------

#### Returns

returns the previously installed error handler.

References CPLSetErrorHandlerEx().

#### 13.2.2.14 CPLErrorHandler CPLSetErrorHandlerEx ( CPLErrorHandler *pfnErrorHandlerNew*, void \* *pUserData* )

Install custom error handle with user's data. This method is essentially CPLSetErrorHandler with an added pointer to pUserData. The pUserData is not returned in the CPLErrorHandler, however, and must be fetched via CPLGetLastErrorUserData

#### Parameters

<i>pfnErrorHandler-New</i>	new error handler function.
<i>pUserData</i>	User data to carry along with the error context.

#### Returns

returns the previously installed error handler.

References CPLDebug().

Referenced by CPLSetErrorHandler().

## 13.3 cpl\_hash\_set.h File Reference

```
#include "cpl_port.h"
```

#### Functions

- **CPLHashSet \* CPLHashSetNew** (CPLHashSetHashFunc fnHashFunc, CPLHashSetEqualFunc fnEqualFunc, CPLHashSetFreeEltFunc fnFreeEltFunc)

- void **CPLHashSetDestroy** (**CPLHashSet** \*set)
- int **CPLHashSetSize** (const **CPLHashSet** \*set)
- void **CPLHashSetForeach** (**CPLHashSet** \*set, CPLHashSetIterEltFunc fnIterFunc, void \*user\_data)
- int **CPLHashSetInsert** (**CPLHashSet** \*set, void \*elt)
- void \* **CPLHashSetLookup** (**CPLHashSet** \*set, const void \*elt)
- int **CPLHashSetRemove** (**CPLHashSet** \*set, const void \*elt)
- unsigned long **CPLHashSetHashPointer** (const void \*elt)
- int **CPLHashSetEqualPointer** (const void \*elt1, const void \*elt2)
- unsigned long **CPLHashSetHashStr** (const void \*pszStr)
- int **CPLHashSetEqualStr** (const void \*pszStr1, const void \*pszStr2)

### 13.3.1 Detailed Description

Hash set implementation.

An hash set is a data structure that holds elements that are unique according to a comparison function. Operations on the hash set, such as insertion, removal or lookup, are supposed to be fast if an efficient "hash" function is provided.

### 13.3.2 Function Documentation

#### 13.3.2.1 void CPLHashSetDestroy ( **CPLHashSet** \* *set* )

Destroys an allocated hash set.

This function also frees the elements if a free function was provided at the creation of the hash set.

Parameters

<i>set</i>	the hash set
------------	--------------

References CPLListDestroy(), \_CPLList::pData, and \_CPLList::pNext.

#### 13.3.2.2 int CPLHashSetEqualPointer ( const void \* *elt1*, const void \* *elt2* )

Equality function for arbitrary pointers

Parameters

<i>elt1</i>	the first arbitrary pointer to compare
<i>elt2</i>	the second arbitrary pointer to compare

Returns

TRUE if the pointers are equal

Referenced by CPLHashSetNew().

#### 13.3.2.3 int CPLHashSetEqualStr ( const void \* *elt1*, const void \* *elt2* )

Equality function for strings

Parameters

<i>elt1</i>	the first string to compare. May be NULL.
<i>elt2</i>	the second string to compare. May be NULL.

**Returns**

TRUE if the strings are equal

#### 13.3.2.4 void CPLHashSetForeach ( CPLHashSet \* *set*, CPLHashSetIterEltFunc *fnIterFunc*, void \* *user\_data* )

Walk through the hash set and runs the provided function on all the elements

This function is provided the *user\_data* argument of CPLHashSetForeach. It must return TRUE to go on the walk through the hash set, or FALSE to make it stop.

Note : the structure of the hash set must *NOT* be modified during the walk.

**Parameters**

<i>set</i>	the hash set.
<i>fnIterFunc</i>	the function called on each element.
<i>user_data</i>	the user data provided to the function.

References \_CPLList::pData, and \_CPLList::psNext.

#### 13.3.2.5 unsigned long CPLHashSetHashPointer ( const void \* *elt* )

Hash function for an arbitrary pointer

**Parameters**

<i>elt</i>	the arbitrary pointer to hash
------------	-------------------------------

**Returns**

the hash value of the pointer

Referenced by CPLHashSetNew().

#### 13.3.2.6 unsigned long CPLHashSetHashStr ( const void \* *elt* )

Hash function for a zero-terminated string

**Parameters**

<i>elt</i>	the string to hash. May be NULL.
------------	----------------------------------

**Returns**

the hash value of the string

#### 13.3.2.7 int CPLHashSetInsert ( CPLHashSet \* *set*, void \* *elt* )

Inserts an element into a hash set.

If the element was already inserted in the hash set, the previous element is replaced by the new element. If a free function was provided, it is used to free the previously inserted element

## Parameters

<i>set</i>	the hash set
<i>elt</i>	the new element to insert in the hash set

## Returns

TRUE if the element was not already in the hash set

References CPLListInsert().

### 13.3.2.8 void\* CPLHashSetLookup ( CPLHashSet \* *set*, const void \* *elt* )

Returns the element found in the hash set corresponding to the element to look up The element must not be modified.

## Parameters

<i>set</i>	the hash set
<i>elt</i>	the element to look up in the hash set

## Returns

the element found in the hash set or NULL

### 13.3.2.9 CPLHashSet\* CPLHashSetNew ( CPLHashSetHashFunc *fnHashFunc*, CPLHashSetEqualFunc *fnEqualFunc*, CPLHashSetFreeEltFunc *fnFreeEltFunc* )

Creates a new hash set

The hash function must return a hash value for the elements to insert. If *fnHashFunc* is NULL, CPLHashSetHashPointer will be used.

The equal function must return if two elements are equal. If *fnEqualFunc* is NULL, CPLHashSetEqualPointer will be used.

The free function is used to free elements inserted in the hash set, when the hash set is destroyed, when elements are removed or replaced. If *fnFreeEltFunc* is NULL, elements inserted into the hash set will not be freed.

## Parameters

<i>fnHashFunc</i>	hash function. May be NULL.
<i>fnEqualFunc</i>	equal function. May be NULL.
<i>fnFreeEltFunc</i>	element free function. May be NULL.

## Returns

a new hash set

References CPLCalloc(), CPLHashSetEqualPointer(), CPLHashSetHashPointer(), and CPLMalloc().

### 13.3.2.10 int CPLHashSetRemove ( CPLHashSet \* *set*, const void \* *elt* )

Removes an element from a hash set

## Parameters

<i>set</i>	the hash set
<i>elt</i>	the new element to remove from the hash set

## Returns

TRUE if the element was in the hash set

References `_CPLList::pData`, and `_CPLList::psNext`.

13.3.2.11 `int CPLHashSetSize ( const CPLHashSet * set )`

Returns the number of elements inserted in the hash set

Note: this is not the internal size of the hash set

## Parameters

<i>set</i>	the hash set
------------	--------------

## Returns

the number of elements in the hash set

## 13.4 cpl\_http.h File Reference

```
#include "cpl_conv.h"
#include "cpl_string.h"
#include "cpl_vsi.h"
```

## Classes

- struct **CPLMimePart**
- struct **CPLHTTPResult**

## Functions

- `int CPLHTTPEnabled (void)`  
*Return if CPLHTTP services can be usefull.*
- `CPLHTTPResult * CPLHTTPFetch (const char *pszURL, char **papszOptions)`  
*Fetch a document from an url and return in a string.*
- `void CPLHTTPCleanup (void)`  
*Cleanup function to call at application termination.*
- `void CPLHTTPDestroyResult (CPLHTTPResult *psResult)`  
*Clean the memory associated with the return value of **CPLHTTPFetch()** (p. ??)*
- `int CPLHTTPParseMultipartMime (CPLHTTPResult *psResult)`  
*Parses a a MIME multipart message.*
- `char * GOA2GetAuthorizationURL (const char *pszScope)`
- `char * GOA2GetRefreshToken (const char *pszAuthToken, const char *pszScope)`
- `char * GOA2GetAccessToken (const char *pszRefreshToken, const char *pszScope)`

### 13.4.1 Detailed Description

Interface for downloading HTTP, FTP documents

### 13.4.2 Function Documentation

#### 13.4.2.1 void CPLHTTPDestroyResult ( CPLHTTPResult \* *psResult* )

Clean the memory associated with the return value of **CPLHTTPFetch()** (p. ??)

##### Parameters

<i>psResult</i>	pointer to the return value of <b>CPLHTTPFetch()</b> (p. ??)
-----------------	--

References CSLDestroy(), CPLHTTPResult::nMimePartCount, CPLHTTPResult::pabyData, CPLMimePart::papsz-Headers, CPLHTTPResult::papszHeaders, CPLHTTPResult::pasMimePart, CPLHTTPResult::pszContentType, and CPLHTTPResult::pszErrBuf.

Referenced by GOA2GetAccessToken(), GOA2GetRefreshToken(), and OGRSpatialReference::importFromUrl().

#### 13.4.2.2 int CPLHTTPEnabled ( void )

Return if CPLHTTP services can be usefull.

Those services depend on GDAL being build with libcurl support.

##### Returns

TRUE if libcurl support is enabled

#### 13.4.2.3 CPLHTTPResult\* CPLHTTPFetch ( const char \* *pszURL*, char \*\* *papszOptions* )

Fetch a document from an url and return in a string.

## Parameters

<i>pszURL</i>	valid URL recognized by underlying download library (libcurl)
<i>papszOptions</i>	option list as a NULL-terminated array of strings. May be NULL. The following options are handled : <ul style="list-style-type: none"> <li>• TIMEOUT=val, where val is in seconds</li> <li>• HEADERS=val, where val is an extra header to use when getting a web page. For example "Accept: application/x-ogcwkt"</li> <li>• HTTPAUTH=[BASIC/NTLM/GSSNEGOTIATE/ANY] to specify an authentication scheme to use.</li> <li>• USERPWD=userid:password to specify a user and password for authentication</li> <li>• POSTFIELDS=val, where val is a nul-terminated string to be passed to the server with a POST request.</li> <li>• PROXY=val, to make requests go through a proxy server, where val is of the form proxy.server.com:port_number</li> <li>• PROXYUSERPWD=val, where val is of the form username:password</li> <li>• PROXYAUTH=[BASIC/NTLM/DIGEST/ANY] to specify an proxy authentication scheme to use.</li> <li>• CUSTOMREQUEST=val, where val is GET, PUT, POST, DELETE, etc.. (GDAL &gt;= 1.9.0)</li> </ul>

Alternatively, if not defined in the papszOptions arguments, the PROXY, PROXYUSERPWD and PROXYAUTH values are searched in the configuration options named GDAL\_HTTP\_PROXY, GDAL\_HTTP\_PROXYUSERPWD and GDAL\_PROXY\_AUTH, as proxy configuration belongs to networking setup and makes more sense at the configuration option level than at the connection level.

## Returns

a CPLHTTPResult\* structure that must be freed by **CPLHTTPDestroyResult()** (p. ??), or NULL if libcurl support is disabled

References CPLCalloc(), CPLDebug(), CPLError(), CPLGetConfigOption(), CPLSetConfigOption(), CPLStrdup(), CSLTestBoolean(), CPLHTTPResult::nDataLen, CPLHTTPResult::nStatus, CPLHTTPResult::papszHeaders, CPLHTTPResult::pszContentType, and CPLHTTPResult::pszErrBuf.

Referenced by GOA2GetAccessToken(), GOA2GetRefreshToken(), and OGRSpatialReference::importFromUrl().

#### 13.4.2.4 int CPLHTTPParseMultipartMime ( CPLHTTPResult \* *psResult* )

Parses a a MIME multipart message.

This function will iterate over each part and put it in a separate element of the pasMimePart array of the provided psResult structure.

## Parameters

<i>psResult</i>	pointer to the return value of <b>CPLHTTPFetch()</b> (p. ??)
-----------------	--

## Returns

TRUE if the message contains MIME multipart message.

References CPLError(), CPLRealloc(), CSLCount(), CSLDestroy(), CPLMimePart::nDataLen, CPLHTTPResult::nDataLen, CPLHTTPResult::nMimePartCount, CPLMimePart::pabyData, CPLHTTPResult::pabyData, CPLMimePart::papszHeaders, CPLHTTPResult::pasMimePart, and CPLHTTPResult::pszContentType.

#### 13.4.2.5 `char* GOA2GetAccessToken ( const char * pszRefreshToken, const char * pszScope )`

Fetch access token using refresh token.

The permanent refresh token is used to fetch a temporary (usually one hour) access token using Google OAuth2 web services.

A CPLError will be reported if the request fails for some reason. Common reasons include the refresh token having been revoked by the user or http connection problems.

##### Parameters

<i>pszRefreshToken</i>	the refresh token from <b>GOA2GetRefreshToken()</b> (p. ??).
<i>pszScope</i>	the scope for which it is valid.

##### Returns

access token, to be freed with CPLFree(), null on failure.

References CPLStringList::AddString(), CPLDebug(), CPLError(), CPLGetConfigOption(), CPLHTTPDestroyResult(), CPLHTTPFetch(), CPLStrdup(), CPLStringList::FetchNameValueDef(), CPLHTTPResult::pabyData, and CPLHTTPResult::pszErrBuf.

#### 13.4.2.6 `char* GOA2GetAuthorizationURL ( const char * pszScope )`

Return authorization url for a given scope.

Returns the URL that a user should visit, and use for authentication in order to get an "auth token" indicating their willingness to use a service.

Note that when the user visits this url they will be asked to login (using a google/gmail/etc) account, and to authorize use of the requested scope for the application "GDAL/OGR". Once they have done so, they will be presented with a lengthy string they should "enter into their application". This is the "auth token" to be passed to **GOA2GetRefreshToken()** (p. ??). The "auth token" can only be used once.

This function should never fail.

##### Parameters

<i>pszScope</i>	the service being requested, not yet URL encoded, such as "https://www.googleapis.com/auth/fusiontables".
-----------------	---

##### Returns

the URL to visit - should be freed with CPLFree().

References CPLEscapeString(), CPLGetConfigOption(), and CPLStrdup().

Referenced by GOA2GetRefreshToken().

#### 13.4.2.7 `char* GOA2GetRefreshToken ( const char * pszAuthToken, const char * pszScope )`

Turn Auth Token into a Refresh Token.

A one time "auth token" provided by the user is turned into a reusable "refresh token" using a google oauth2 web service.

A CPLError will be reported if the translation fails for some reason. Common reasons include the auth token already having been used before, it not being appropriate for the passed scope and configured client api or http connection problems. NULL is returned on error.



## Parameters

<i>pszAuthToken</i>	the authorization token from the user.
<i>pszScope</i>	the scope for which it is valid.

## Returns

refresh token, to be freed with CPLFree(), null on failure.

References CPLStringList::AddString(), CPLDebug(), CPLError(), CPLGetConfigOption(), CPLHTTPDestroyResult(), CPLHTTPFetch(), CPLStrdup(), CPLStringList::FetchNameValueDef(), GOA2GetAuthorizationURL(), CPLHTTPResult::pabyData, and CPLHTTPResult::pszErrBuf.

## 13.5 cpl\_list.h File Reference

```
#include "cpl_port.h"
```

## Classes

- struct **\_CPLList**

## Typedefs

- typedef struct **\_CPLList CPLList**

## Functions

- **CPLList \* CPLListAppend** (**CPLList** \*psList, void \*pData)
- **CPLList \* CPLListInsert** (**CPLList** \*psList, void \*pData, int nPosition)
- **CPLList \* CPLListGetLast** (**CPLList** \*psList)
- **CPLList \* CPLListGet** (**CPLList** \*psList, int nPosition)
- int **CPLListCount** (**CPLList** \*psList)
- **CPLList \* CPLListRemove** (**CPLList** \*psList, int nPosition)
- void **CPLListDestroy** (**CPLList** \*psList)
- **CPLList \* CPLListGetNext** (**CPLList** \*psElement)
- void \* **CPLListGetData** (**CPLList** \*psElement)

### 13.5.1 Detailed Description

Simplest list implementation. List contains only pointers to stored objects, not objects itself. All operations regarding allocation and freeing memory for objects should be performed by the caller.

### 13.5.2 Typedef Documentation

#### 13.5.2.1 typedef struct **\_CPLList CPLList**

List element structure.

### 13.5.3 Function Documentation

#### 13.5.3.1 `CPLList*` `CPLListAppend ( CPLList * psList, void * pData )`

Append an object list and return a pointer to the modified list. If the input list is NULL, then a new list is created.

## Parameters

<i>psList</i>	pointer to list head.
<i>pData</i>	pointer to inserted data object. May be NULL.

## Returns

pointer to the head of modified list.

References CPLListGetLast(), CPLMalloc(), \_CPLList::pData, and \_CPLList::pNext.

Referenced by CPLListInsert().

### 13.5.3.2 int CPLListCount ( CPLList \* *psList* )

Return the number of elements in a list.

## Parameters

<i>psList</i>	pointer to list head.
---------------	-----------------------

## Returns

number of elements in a list.

References \_CPLList::pNext.

Referenced by CPLListInsert().

### 13.5.3.3 void CPLListDestroy ( CPLList \* *psList* )

Destroy a list. Caller responsible for freeing data objects contained in list elements.

## Parameters

<i>psList</i>	pointer to list head.
---------------	-----------------------

References \_CPLList::pNext.

Referenced by CPLHashSetDestroy().

### 13.5.3.4 CPLList\* CPLListGet ( CPLList \* *psList*, int *nPosition* )

Return the pointer to the specified element in a list.

## Parameters

<i>psList</i>	pointer to list head.
<i>nPosition</i>	the index of the element in the list, 0 being the first element

## Returns

pointer to the specified element in a list.

References \_CPLList::pNext.

### 13.5.3.5 void\* CPLListGetData ( CPLList \* *psElement* )

Return pointer to the data object contained in given list element.

## Parameters

<i>psElement</i>	pointer to list element.
------------------	--------------------------

## Returns

pointer to the data object contained in given list element.

References `_CPLList::pData`.

### 13.5.3.6 `CPLList*` `CPLListGetLast ( CPLList * psList )`

Return the pointer to last element in a list.

## Parameters

<i>psList</i>	pointer to list head.
---------------	-----------------------

## Returns

pointer to last element in a list.

References `_CPLList::psNext`.

Referenced by `CPLListAppend()`, and `CPLListInsert()`.

### 13.5.3.7 `CPLList*` `CPLListGetNext ( CPLList * psElement )`

Return the pointer to next element in a list.

## Parameters

<i>psElement</i>	pointer to list element.
------------------	--------------------------

## Returns

pointer to the list element preceded by the given element.

References `_CPLList::psNext`.

### 13.5.3.8 `CPLList*` `CPLListInsert ( CPLList * psList, void * pData, int nPosition )`

Insert an object into list at specified position (zero based). If the input list is NULL, then a new list is created.

## Parameters

<i>psList</i>	pointer to list head.
<i>pData</i>	pointer to inserted data object. May be NULL.
<i>nPosition</i>	position number to insert an object.

## Returns

pointer to the head of modified list.

References `CPLListAppend()`, `CPLListCount()`, `CPLListGetLast()`, `CPLMalloc()`, `_CPLList::pData`, and `_CPLList::psNext`.

Referenced by `CPLHashSetInsert()`.

#### 13.5.3.9 CPLLlist\* CPLLlistRemove ( CPLLlist \* *psList*, int *nPosition* )

Remove the element from the specified position (zero based) in a list. Data object contained in removed element must be freed by the caller first.

## Parameters

<i>psList</i>	pointer to list head.
<i>nPosition</i>	position number to delet an element.

## Returns

pointer to the head of modified list.

References `_CPLList::psNext`.

## 13.6 cpl\_minixml.h File Reference

```
#include "cpl_port.h"
```

## Classes

- struct **CPLXMLNode**

## Typedefs

- typedef struct **CPLXMLNode** **CPLXMLNode**

## Enumerations

- enum **CPLXMLNodeType** {  
**CXT\_Element** = 0, **CXT\_Text** = 1, **CXT\_Attribute** = 2, **CXT\_Comment** = 3,  
**CXT\_Literal** = 4 }

## Functions

- **CPLXMLNode \* CPLParseXMLString** (const char \*)  
*Parse an XML string into tree form.*
- void **CPLDestroyXMLNode** (**CPLXMLNode** \*)  
*Destroy a tree.*
- **CPLXMLNode \* CPLGetXMLNode** (**CPLXMLNode** \*poRoot, const char \*pszPath)  
*Find node by path.*
- **CPLXMLNode \* CPLSearchXMLNode** (**CPLXMLNode** \*poRoot, const char \*pszTarget)  
*Search for a node in document.*
- const char \* **CPLGetXMLValue** (**CPLXMLNode** \*poRoot, const char \*pszPath, const char \*pszDefault)  
*Fetch element/attribute value.*
- **CPLXMLNode \* CPLCreateXMLNode** (**CPLXMLNode** \*poParent, **CPLXMLNodeType** eType, const char \*pszText)  
*Create an document tree item.*
- char \* **CPLSerializeXMLTree** (const **CPLXMLNode** \*psNode)  
*Convert tree into string document.*
- void **CPLAddXMLChild** (**CPLXMLNode** \*psParent, **CPLXMLNode** \*psChild)  
*Add child node to parent.*
- int **CPLRemoveXMLChild** (**CPLXMLNode** \*psParent, **CPLXMLNode** \*psChild)  
*Remove child node from parent.*

- void **CPLAddXMLSibling** (**CPLXMLNode** \*psOlderSibling, **CPLXMLNode** \*psNewSibling)  
*Add new sibling.*
- **CPLXMLNode** \* **CPLCreateXMLElementAndValue** (**CPLXMLNode** \*psParent, const char \*pszName, const char \*pszValue)  
*Create an element and text value.*
- **CPLXMLNode** \* **CPLCloneXMLTree** (**CPLXMLNode** \*psTree)  
*Copy tree.*
- int **CPLSetXMLValue** (**CPLXMLNode** \*psRoot, const char \*pszPath, const char \*pszValue)  
*Set element value by path.*
- void **CPLStripXMLNamespace** (**CPLXMLNode** \*psRoot, const char \*pszNameSpace, int bRecurse)  
*Strip indicated namespaces.*
- void **CPLCleanXMLElementName** (char \*)  
*Make string into safe XML token.*
- **CPLXMLNode** \* **CPLParseXMLFile** (const char \*pszFilename)  
*Parse XML file into tree.*
- int **CPLSerializeXMLTreeToFile** (const **CPLXMLNode** \*psTree, const char \*pszFilename)  
*Write document tree to a file.*

### 13.6.1 Detailed Description

Definitions for CPL mini XML Parser/Serializer.

### 13.6.2 Typedef Documentation

#### 13.6.2.1 typedef struct **CPLXMLNode** **CPLXMLNode**

Document node structure.

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

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

### 13.6.3 Enumeration Type Documentation

#### 13.6.3.1 enum **CPLXMLNodeType**

Enumerator

- CXT\_Element** Node is an element
- CXT\_Text** Node is a raw text value
- CXT\_Attribute** Node is attribute
- CXT\_Comment** Node is an XML comment.
- CXT\_Literal** Node is a special literal

### 13.6.4 Function Documentation

#### 13.6.4.1 void CPLAddXMLChild ( CPLXMLNode \* *psParent*, CPLXMLNode \* *psChild* )

Add child node to parent.

The passed child is added to the list of children of the indicated parent. Normally the child is added at the end of the parents child list, but attributes (CXT\_Attribute) will be inserted after any other attributes but before any other element type. Ownership of the child node is effectively assumed by the parent node. If the child has siblings (it's psNext is not NULL) they will be trimmed, but if the child has children they are carried with it.

##### Parameters

<i>psParent</i>	the node to attach the child to. May not be NULL.
<i>psChild</i>	the child to add to the parent. May not be NULL. Should not be a child of any other parent.

References CXT\_Attribute, CPLXMLNode::eType, CPLXMLNode::psChild, and CPLXMLNode::psNext.

#### 13.6.4.2 void CPLAddXMLSibling ( CPLXMLNode \* *psOlderSibling*, CPLXMLNode \* *psNewSibling* )

Add new sibling.

The passed psNewSibling is added to the end of siblings of the psOlderSibling node. That is, it is added to the end of the psNext chain. There is no special handling if psNewSibling is an attribute. If this is required, use **CPLAddXMLChild()** (p. ??).

##### Parameters

<i>psOlderSibling</i>	the node to attach the sibling after.
<i>psNewSibling</i>	the node to add at the end of psOlderSiblings psNext chain.

References CPLXMLNode::psNext.

#### 13.6.4.3 void CPLCleanXMLElementName ( char \* *pszTarget* )

Make string into safe XML token.

Modifies a string in place to try and make it into a legal XML token that can be used as an element name. This is accomplished by changing any characters not legal in a token into an underscore.

NOTE: This function should implement the rules in section 2.3 of <http://www.w3.org/TR/xml11/> but it doesn't yet do that properly. We only do a rough approximation of that.

##### Parameters

<i>pszTarget</i>	the string to be adjusted. It is altered in place.
------------------	--

#### 13.6.4.4 CPLXMLNode\* CPLCloneXMLTree ( CPLXMLNode \* *psTree* )

Copy tree.

Creates a deep copy of a **CPLXMLNode** (p. ??) tree.

##### Parameters

<i>psTree</i>	the tree to duplicate.
---------------	------------------------



**Returns**

a copy of the whole tree.

References CPLCreateXMLNode(), CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

#### 13.6.4.5 **CPLXMLNode\*** CPLCreateXMLElementAndValue ( **CPLXMLNode** \* *psParent*, **const char** \* *pszName*, **const char** \* *pszValue* )

Create an element and text value.

This is function is a convenient short form for:

```
*   CPLXMLNode *psTextNode;
*   CPLXMLNode *psElementNode;
*
*   psElementNode = CPLCreateXMLNode( psParent, CXT_Element, pszName );
*   psTextNode = CPLCreateXMLNode( psElementNode, CXT_Text, pszValue );
*
*   return psElementNode;
*
```

It creates a CXT\_Element node, with a CXT\_Text child, and attaches the element to the passed parent.

**Parameters**

<i>psParent</i>	the parent node to which the resulting node should be attached. May be NULL to keep as freestanding.
<i>pszName</i>	the element name to create.
<i>pszValue</i>	the text to attach to the element. Must not be NULL.

**Returns**

the pointer to the new element node.

References CPLCreateXMLNode(), CXT\_Element, and CXT\_Text.

#### 13.6.4.6 **CPLXMLNode\*** CPLCreateXMLNode ( **CPLXMLNode** \* *poParent*, **CPLXMLNodeType** *eType*, **const char** \* *pszText* )

Create an document tree item.

Create a single **CPLXMLNode** (p. ??) object with the desired value and type, and attach it as a child of the indicated parent.

**Parameters**

<i>poParent</i>	the parent to which this node should be attached as a child. May be NULL to keep as free standing.
<i>eType</i>	the type of the newly created node
<i>pszText</i>	the value of the newly created node

**Returns**

the newly created node, now owned by the caller (or parent node).

References CPLCalloc(), CPLStrdup(), CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

Referenced by CPLCloneXMLTree(), CPLCreateXMLElementAndValue(), and CPLSetXMLValue().

#### 13.6.4.7 void CPLDestroyXMLNode ( CPLXMLNode \* *psNode* )

Destroy a tree.

This function frees resources associated with a **CPLXMLNode** (p. ??) and all its children nodes.

##### Parameters

<i>psNode</i>	the tree to free.
---------------	-------------------

References CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

Referenced by CPLParseXMLString(), OGRSpatialReference::exportToXML(), OGRSpatialReference::importFromXML(), and OGR\_G\_CreateFromGML().

#### 13.6.4.8 CPLXMLNode\* CPLGetXMLNode ( CPLXMLNode \* *psRoot*, const char \* *pszPath* )

Find node by path.

Searches the document or subdocument indicated by psRoot for an element (or attribute) with the given path. The path should consist of a set of element names separated by dots, not including the name of the root element (psRoot). If the requested element is not found NULL is returned.

Attribute names may only appear as the last item in the path.

The search is done from the root nodes children, but all intermediate nodes in the path must be specified. Searching for "name" would only find a name element or attribute if it is a direct child of the root, not at any level in the subdocument.

If the pszPath is prefixed by "=" then the search will begin with the root node, and it's siblings, instead of the root nodes children. This is particularly useful when searching within a whole document which is often prefixed by one or more "junk" nodes like the <?xml> declaration.

##### Parameters

<i>psRoot</i>	the subtree in which to search. This should be a node of type CXT_Element. NULL is safe.
<i>pszPath</i>	the list of element names in the path (dot separated).

##### Returns

the requested element node, or NULL if not found.

References CSLDestroy(), CXT\_Text, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

Referenced by CPLGetXMLValue().

#### 13.6.4.9 const char\* CPLGetXMLValue ( CPLXMLNode \* *psRoot*, const char \* *pszPath*, const char \* *pszDefault* )

Fetch element/attribute value.

Searches the document for the element/attribute value associated with the path. The corresponding node is internally found with **CPLGetXMLNode()** (p. ??) (see there for details on path handling). Once found, the value is considered to be the first CXT\_Text child of the node.

If the attribute/element search fails, or if the found node has not value then the passed default value is returned.

The returned value points to memory within the document tree, and should not be altered or freed.

##### Parameters

<i>psRoot</i>	the subtree in which to search. This should be a node of type CXT_Element. NULL is safe.
<i>pszPath</i>	the list of element names in the path (dot separated). An empty path means get the value of the psRoot node.
<i>pszDefault</i>	the value to return if a corresponding value is not found, may be NULL.

**Returns**

the requested value or pszDefault if not found.

References CPLGetXMLNode(), CXT\_Attribute, CXT\_Element, CXT\_Text, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

**13.6.4.10 CPLXMLNode\* CPLParseXMLFile ( const char \* *pszFilename* )**

Parse XML file into tree.

The named file is opened, loaded into memory as a big string, and parsed with **CPLParseXMLString()** (p. ??). Errors in reading the file or parsing the XML will be reported by **CPLERROR()** (p. ??).

The "large file" API is used, so XML files can come from virtualized files.

**Parameters**

<i>pszFilename</i>	the file to open.
--------------------	-------------------

**Returns**

NULL on failure, or the document tree on success.

References CPLERROR(), CPLParseXMLString(), VSIFCloseL(), VSIFOpenL(), VSIFReadL(), VSIFSeekL(), and VSIFTellL().

**13.6.4.11 CPLXMLNode\* CPLParseXMLString ( const char \* *pszString* )**

Parse an XML string into tree form.

The passed document is parsed into a **CPLXMLNode** (p. ??) tree representation. If the document is not well formed XML then NULL is returned, and errors are reported via **CPLERROR()** (p. ??). No validation beyond wellformedness is done. The **CPLParseXMLFile()** (p. ??) convenience function can be used to parse from a file.

The returned document tree is owned by the caller and should be freed with **CPLDestroyXMLNode()** (p. ??) when no longer needed.

If the document has more than one "root level" element then those after the first will be attached to the first as siblings (via the psNext pointers) even though there is no common parent. A document with no XML structure (no angle brackets for instance) would be considered well formed, and returned as a single CXT\_Text node.

**Parameters**

<i>pszString</i>	the document to parse.
------------------	------------------------

**Returns**

parsed tree or NULL on error.

References CPLDestroyXMLNode(), CPLERROR(), CPLERRORReset(), CPLGetLastErrorType(), CXT\_Attribute, CXT\_Comment, CXT\_Element, CXT\_Literal, CXT\_Text, and CPLXMLNode::pszValue.

Referenced by CPLParseXMLFile(), OGRSpatialReference::importFromXML(), and OGR\_G\_CreateFromGML().

#### 13.6.4.12 int CPLRemoveXMLChild ( CPLXMLNode \* *psParent*, CPLXMLNode \* *psChild* )

Remove child node from parent.

The passed child is removed from the child list of the passed parent, but the child is not destroyed. The child retains ownership of it's own children, but is cleanly removed from the child list of the parent.

##### Parameters

<i>psParent</i>	the node to the child is attached to.
<i>psChild</i>	the child to remove.

##### Returns

TRUE on success or FALSE if the child was not found.

References CPLXMLNode::psChild, and CPLXMLNode::psNext.

#### 13.6.4.13 CPLXMLNode\* CPLSearchXMLNode ( CPLXMLNode \* *psRoot*, const char \* *pszElement* )

Search for a node in document.

Searches the children (and potentially siblings) of the documented passed in for the named element or attribute. To search following siblings as well as children, prefix the pszElement name with an equal sign. This function does an in-order traversal of the document tree. So it will first match against the current node, then it's first child, that child's first child, and so on.

Use **CPLGetXMLNode()** (p. ??) to find a specific child, or along a specific node path.

##### Parameters

<i>psRoot</i>	the subtree to search. This should be a node of type CXT_Element. NULL is safe.
<i>pszElement</i>	the name of the element or attribute to search for.

##### Returns

The matching node or NULL on failure.

References CXT\_Attribute, CXT\_Element, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

#### 13.6.4.14 char\* CPLSerializeXMLTree ( const CPLXMLNode \* *psNode* )

Convert tree into string document.

This function converts a **CPLXMLNode** (p. ??) tree representation of a document into a flat string representation. White space indentation is used visually preserve the tree structure of the document. The returned document becomes owned by the caller and should be freed with CPLFree() when no longer needed.

##### Parameters

<i>psNode</i>	the node to serialize.
---------------	------------------------

##### Returns

the document on success or NULL on failure.

References CPLMalloc(), and CPLXMLNode::psNext.

Referenced by CPLSerializeXMLTreeToFile(), and OGRSpatialReference::exportToXML().

#### 13.6.4.15 int CPLSerializeXMLTreeToFile ( const CPLXMLNode \* *psTree*, const char \* *pszFilename* )

Write document tree to a file.

The passed document tree is converted into one big string (with **CPLSerializeXMLTree()** (p. ??)) and then written to the named file. Errors writing the file will be reported by **CPLError()** (p. ??). The source document tree is not altered. If the output file already exists it will be overwritten.

##### Parameters

<i>psTree</i>	the document tree to write.
<i>pszFilename</i>	the name of the file to write to.

##### Returns

TRUE on success, FALSE otherwise.

References CPLError(), CPLSerializeXMLTree(), VSIFCloseL(), VSIFOpenL(), and VSIFWriteL().

#### 13.6.4.16 int CPLSetXMLValue ( CPLXMLNode \* *psRoot*, const char \* *pszPath*, const char \* *pszValue* )

Set element value by path.

Find (or create) the target element or attribute specified in the path, and assign it the indicated value.

Any path elements that do not already exist will be created. The target nodes value (the first CXT\_Text child) will be replaced with the provided value.

If the target node is an attribute instead of an element, the name should be prefixed with a #.

Example: CPLSetXMLValue( "Citation.Id.Description", "DOQ dataset" ); CPLSetXMLValue( "Citation.Id-Description.#name", "doq" );

##### Parameters

<i>psRoot</i>	the subdocument to be updated.
<i>pszPath</i>	the dot seperated path to the target element/attribute.
<i>pszValue</i>	the text value to assign.

##### Returns

TRUE on success.

References CPLCreateXMLNode(), CPLStrdup(), CSLDestroy(), CXT\_Attribute, CXT\_Element, CXT\_Text, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

#### 13.6.4.17 void CPLStripXMLNamespace ( CPLXMLNode \* *psRoot*, const char \* *pszNamespace*, int *bRecurse* )

Strip indicated namespaces.

The subdocument (psRoot) is recursively examined, and any elements with the indicated namespace prefix will have the namespace prefix stripped from the element names. If the passed namespace is NULL, then all namespace prefixes will be stripped.

Nodes other than elements should remain unaffected. The changes are made "in place", and should not alter any node locations, only the pszValue field of affected nodes.

##### Parameters

<i>psRoot</i>	the document to operate on.
<i>pszNamespace</i>	the name space prefix (not including colon), or NULL.
<i>bRecurse</i>	TRUE to recurse over whole document, or FALSE to only operate on the passed node.

References CXT\_Attribute, CXT\_Element, CPLXMLNode::eType, CPLXMLNode::psChild, CPLXMLNode::psNext, and CPLXMLNode::pszValue.

Referenced by OGRSpatialReference::importFromXML().

## 13.7 cpl\_odbc.h File Reference

```
#include "cpl_port.h"
#include <sql.h>
#include <sqlext.h>
#include <odbcinst.h>
#include "cpl_string.h"
```

### Classes

- class **CPLODBCDriverInstaller**
- class **CPLODBCSession**
- class **CPLODBCStatement**

### 13.7.1 Detailed Description

ODBC Abstraction Layer (C++).

## 13.8 cpl\_port.h File Reference

```
#include "cpl_config.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdarg.h>
#include <string.h>
#include <ctype.h>
#include <limits.h>
#include <time.h>
#include <errno.h>
#include <locale.h>
```

### Macros

- #define **CPL\_LSBINT16PTR(x)** ((\*(GByte\*)(x)) | (\*(GByte\*)((x)+1)) << 8)
- #define **CPL\_LSBINT32PTR(x)**
- #define **CPL\_LSBSINT16PTR(x)** ((GInt16) CPL\_LSBINT16PTR(x))
- #define **CPL\_LSBUINT16PTR(x)** ((GUInt16)CPL\_LSBINT16PTR(x))
- #define **CPL\_LSBSINT32PTR(x)** ((GInt32) CPL\_LSBINT32PTR(x))
- #define **CPL\_LSBUINT32PTR(x)** ((GUInt32)CPL\_LSBINT32PTR(x))

### 13.8.1 Detailed Description

Core portability definitions for CPL.

### 13.8.2 Macro Definition Documentation

**13.8.2.1** `#define CPL_LSBINT16PTR( x ) ((*(GByte*)(x)) | (*(GByte*)((x)+1)) << 8)`

Return a Int16 from the 2 bytes ordered in LSB order at address x

**13.8.2.2** `#define CPL_LSBINT32PTR( x )`

**Value:**

```
((*(GByte*)(x)) | ((*(GByte*)((x)+1)) << 8) | \
  ((*(GByte*)((x)+2)) << 16) | ((*(GByte*)((x)+3)) << 24))
```

Return a Int32 from the 4 bytes ordered in LSB order at address x

**13.8.2.3** `#define CPL_LSBSINT16PTR( x ) ((GInt16)CPL_LSBINT16PTR(x))`

Return a signed Int16 from the 2 bytes ordered in LSB order at address x

**13.8.2.4** `#define CPL_LSBSINT32PTR( x ) ((GInt32)CPL_LSBINT32PTR(x))`

Return a signed Int32 from the 4 bytes ordered in LSB order at address x

**13.8.2.5** `#define CPL_LSBUINT16PTR( x ) ((GUInt16)CPL_LSBINT16PTR(x))`

Return a unsigned Int16 from the 2 bytes ordered in LSB order at address x

**13.8.2.6** `#define CPL_LSBUINT32PTR( x ) ((GUInt32)CPL_LSBINT32PTR(x))`

Return a unsigned Int32 from the 4 bytes ordered in LSB order at address x

## 13.9 cpl\_quad\_tree.h File Reference

```
#include "cpl_port.h"
```

### Classes

- struct **CPLRectObj**

### Functions

- **CPLQuadTree** \* **CPLQuadTreeCreate** (const **CPLRectObj** \*pGlobalBounds, CPLQuadTreeGetBounds-  
Func pfnGetBounds)
- void **CPLQuadTreeDestroy** (**CPLQuadTree** \*hQuadtree)
- void **CPLQuadTreeSetBucketCapacity** (**CPLQuadTree** \*hQuadtree, int nBucketCapacity)

- int **CPLQuadTreeGetAdvisedMaxDepth** (int nExpectedFeatures)
- void **CPLQuadTreeSetMaxDepth** (**CPLQuadTree** \*hQuadtree, int nMaxDepth)
- void **CPLQuadTreeInsert** (**CPLQuadTree** \*hQuadtree, void \*hFeature)
- void \*\* **CPLQuadTreeSearch** (const **CPLQuadTree** \*hQuadtree, const **CPLRectObj** \*pAoi, int \*pnFeatureCount)
- void **CPLQuadTreeForeach** (const **CPLQuadTree** \*hQuadtree, CPLQuadTreeForeachFunc pfnForeach, void \*pUserData)

### 13.9.1 Detailed Description

Quad tree implementation.

A quadtree is a tree data structure in which each internal node has up to four children. Quadtrees are most often used to partition a two dimensional space by recursively subdividing it into four quadrants or regions

### 13.9.2 Function Documentation

#### 13.9.2.1 **CPLQuadTree\*** CPLQuadTreeCreate ( const **CPLRectObj** \* *pGlobalBounds*, CPLQuadTreeGetBoundsFunc *pfnGetBounds* )

Create a new quadtree

Parameters

<i>pGlobalBounds</i>	a pointer to the global extent of all the elements that will be inserted
<i>pfnGetBounds</i>	a user provided function to get the bounding box of the inserted elements

Returns

a newly allocated quadtree

References CPLMalloc().

#### 13.9.2.2 void **CPLQuadTreeDestroy** ( **CPLQuadTree** \* *hQuadTree* )

Destroy a quadtree

Parameters

<i>hQuadTree</i>	the quad tree to destroy
------------------	--------------------------

#### 13.9.2.3 void **CPLQuadTreeForeach** ( const **CPLQuadTree** \* *hQuadTree*, CPLQuadTreeForeachFunc *pfnForeach*, void \* *pUserData* )

Walk through the quadtree and runs the provided function on all the elements

This function is provided with the user\_data argument of pfnForeach. It must return TRUE to go on the walk through the hash set, or FALSE to make it stop.

Note : the structure of the quadtree must *NOT* be modified during the walk.

Parameters

<i>hQuadTree</i>	the quad tree
------------------	---------------



<i>pfnForeach</i>	the function called on each element.
<i>pUserData</i>	the user data provided to the function.

#### 13.9.2.4 int CPLQuadTreeGetAdvisedMaxDepth ( int *nExpectedFeatures* )

Returns the optimal depth of a quadtree to hold *nExpectedFeatures*

Parameters

<i>nExpectedFeatures</i>	the expected maximum number of elements to be inserted
--------------------------	--

Returns

the optimal depth of a quadtree to hold *nExpectedFeatures*

References CPLDebug().

#### 13.9.2.5 void CPLQuadTreeInsert ( CPLQuadTree \* *hQuadTree*, void \* *hFeature* )

Insert a feature into a quadtree

Parameters

<i>hQuadTree</i>	the quad tree
<i>hFeature</i>	the feature to insert

#### 13.9.2.6 void\*\* CPLQuadTreeSearch ( const CPLQuadTree \* *hQuadTree*, const CPLRectObj \* *pAoi*, int \* *pnFeatureCount* )

Returns all the elements inserted whose bounding box intersects the provided area of interest

Parameters

<i>hQuadTree</i>	the quad tree
<i>pAoi</i>	the pointer to the area of interest
<i>pnFeatureCount</i>	the user data provided to the function.

Returns

an array of features that must be freed with CPLFree

#### 13.9.2.7 void CPLQuadTreeSetBucketCapacity ( CPLQuadTree \* *hQuadTree*, int *nBucketCapacity* )

Set the maximum capacity of a node of a quadtree. The default value is 8. Note that the maximum capacity will only be honoured if the features inserted have a point geometry. Otherwise it may be exceeded.

Parameters

<i>hQuadTree</i>	the quad tree
<i>nBucketCapacity</i>	the maximum capacity of a node of a quadtree

### 13.9.2.8 void CPLQuadTreeSetMaxDepth ( CPLQuadTree \* hQuadTree, int nMaxDepth )

Set the maximum depth of a quadtree. By default, quad trees have no maximum depth, but a maximum bucket capacity.

#### Parameters

<i>hQuadTree</i>	the quad tree
<i>nMaxDepth</i>	the maximum depth allowed

## 13.10 cpl\_string.h File Reference

```
#include "cpl_vsi.h"
#include "cpl_error.h"
#include "cpl_conv.h"
#include <string>
```

### Classes

- class **CPLString**  
*Convenient string class based on std::string.*
- class **CPLStringList**  
*String list class designed around our use of C "char\*" string lists.*

### Functions

- int **CSLCount** (char \*\*papszStrList)
- void **CSLDestroy** (char \*\*papszStrList)
- char \*\* **CSLDuplicate** (char \*\*papszStrList)
- char \*\* **CSLMerge** (char \*\*papszOrig, char \*\*papszOverride)  
*Merge two lists.*
- char \*\* **CSLTokenizeString2** (const char \*pszString, const char \*pszDelimiter, int nCSLTFlags)
- char \*\* **CSLLoad** (const char \*pszFname)
- char \*\* **CSLLoad2** (const char \*pszFname, int nMaxLines, int nMaxCols, char \*\*papszOptions)
- int **CSLFindString** (char \*\*, const char \*)
- int **CSLPartialFindString** (char \*\*papszHaystack, const char \*pszNeedle)
- int **CSLFindName** (char \*\*papszStrList, const char \*pszName)
- int **CSLTestBoolean** (const char \*pszValue)
- const char \* **CPLParseNameValue** (const char \*pszNameValue, char \*\*ppszKey)
- char \*\* **CSLSetNameValue** (char \*\*papszStrList, const char \*pszName, const char \*pszValue)
- void **CSLSetNameValueSeparator** (char \*\*papszStrList, const char \*pszSeparator)
- char \* **CPLEscapeString** (const char \*pszString, int nLength, int nScheme)
- char \* **CPLUnescapeString** (const char \*pszString, int \*pnLength, int nScheme)
- char \* **CPLBinaryToHex** (int nBytes, const GByte \*pabyData)
- GByte \* **CPLHexToBinary** (const char \*pszHex, int \*pnBytes)
- CPLValueType **CPLGetValueType** (const char \*pszValue)
- size\_t **CPLStrlcpy** (char \*pszDest, const char \*pszSrc, size\_t nDestSize)
- size\_t **CPLStrlcat** (char \*pszDest, const char \*pszSrc, size\_t nDestSize)
- size\_t **CPLStrnlen** (const char \*pszStr, size\_t nMaxLen)

- int **CPLEncodingCharSize** (const char \*pszEncoding)
- char \* **CPLRecode** (const char \*pszSource, const char \*pszSrcEncoding, const char \*pszDstEncoding)
- char \* **CPLRecodeFromWChar** (const wchar\_t \*pwszSource, const char \*pszSrcEncoding, const char \*pszDstEncoding)
- wchar\_t \* **CPLRecodeToWChar** (const char \*pszSource, const char \*pszSrcEncoding, const char \*pszDstEncoding)
- int **CPLIsUTF8** (const char \*pabyData, int nLen)
- char \* **CPLForceToASCII** (const char \*pabyData, int nLen, char chReplacementChar)
- **CPLString CPLURLGetValue** (const char \*pszURL, const char \*pszKey)
- **CPLString CPLURLAddKVP** (const char \*pszURL, const char \*pszKey, const char \*pszValue)

### 13.10.1 Detailed Description

Various convenience functions for working with strings and string lists.

A StringList is just an array of strings with the last pointer being NULL. An empty StringList may be either a NULL pointer, or a pointer to a pointer memory location with a NULL value.

A common convention for StringLists is to use them to store name/value lists. In this case the contents are treated like a dictionary of name/value pairs. The actual data is formatted with each string having the format "<name>:<value>" (though "=" is also an acceptable separator). A number of the functions in the file operate on name/value style string lists (such as **CSLSetNameValue()** (p. ??), and **CSLFetchNameValue()**).

To some extent the **CPLStringList** (p. ??) C++ class can be used to abstract managing string lists a bit but still be able to return them from C functions.

### 13.10.2 Function Documentation

#### 13.10.2.1 char\* CPLBinaryToHex ( int nBytes, const GByte \* pabyData )

Binary to hexadecimal translation.

##### Parameters

<i>nBytes</i>	number of bytes of binary data in pabyData.
<i>pabyData</i>	array of data bytes to translate.

##### Returns

hexadecimal translation, zero terminated. Free with CPLFree().

References CPLMalloc().

Referenced by OGRFeature::GetFieldAsString().

#### 13.10.2.2 int CPLEncodingCharSize ( const char \* pszEncoding )

Return bytes per character for encoding.

This function returns the size in bytes of the smallest character in this encoding. For fixed width encodings (ASCII, UCS-2, UCS-4) this is straight forward. For encodings like UTF8 and UTF16 which represent some characters as a sequence of atomic character sizes the function still returns the atomic character size (1 for UTF8, 2 for UTF16).

This function will return the correct value for well known encodings with corresponding CPL\_ENC\_ values. It may not return the correct value for other encodings even if they are supported by the underlying iconv or windows transliteration services. Hopefully it will improve over time.

## Parameters

<i>pszEncoding</i>	the name of the encoding.
--------------------	---------------------------

## Returns

the size of a minimal character in bytes or -1 if the size is unknown.

### 13.10.2.3 char\* CPLEscapeString ( const char \* *pszInput*, int *nLength*, int *nScheme* )

Apply escaping to string to preserve special characters.

This function will "escape" a variety of special characters to make the string suitable to embed within a string constant or to write within a text stream but in a form that can be reconstituted to it's original form. The escaping will even preserve zero bytes allowing preservation of raw binary data.

CPLES\_BackslashQuotable(0): This scheme turns a binary string into a form suitable to be placed within double quotes as a string constant. The backslash, quote, '\0' and newline characters are all escaped in the usual C style.

CPLES\_XML(1): This scheme converts the '<', '>', '"' and '&' characters into their XML/HTML equivalent (<, >, " and &) making a string safe to embed as CDATA within an XML element. The '\0' is not escaped and should not be included in the input.

CPLES\_URL(2): Everything except alphanumerics and the underscore are converted to a percent followed by a two digit hex encoding of the character (leading zero supplied if needed). This is the mechanism used for encoding values to be passed in URLs.

CPLES\_SQL(3): All single quotes are replaced with two single quotes. Suitable for use when constructing literal values for SQL commands where the literal will be enclosed in single quotes.

CPLES\_CSV(4): If the values contains commas, semicolons, tabs, double quotes, or newlines it placed in double quotes, and double quotes in the value are doubled. Suitable for use when constructing field values for .csv files. Note that **CPLUnescapeString()** (p. ??) currently does not support this format, only **CPLEscapeString()** (p. ??). See `cpl_csv.cpp` for csv parsing support.

## Parameters

<i>pszInput</i>	the string to escape.
<i>nLength</i>	The number of bytes of data to preserve. If this is -1 the <code>strlen(pszString)</code> function will be used to compute the length.
<i>nScheme</i>	the encoding scheme to use.

## Returns

an escaped, zero terminated string that should be freed with `CPLFree()` when no longer needed.

References `CPLError()`, `CPLMalloc()`, and `CPLStrdup()`.

Referenced by `GOA2GetAuthorizationURL()`, and `OGRGeocode()`.

### 13.10.2.4 char\* CPLForceToASCII ( const char \* *pabyData*, int *nLen*, char *chReplacementChar* )

Return a new string that is made only of ASCII characters. If non-ASCII characters are found in the input string, they will be replaced by the provided replacement character.

## Parameters

<i>pabyData</i>	input string to test
<i>nLen</i>	length of the input string, or -1 if the function must compute the string length. In which case it must be null terminated.
<i>chReplacement-Char</i>	character which will be used when the input stream contains a non ASCII character. Must be valid ASCII !

**Returns**

a new string that must be freed with CPLFree().

**Since**

GDAL 1.7.0

References CPLMalloc().

**13.10.2.5 CPLValueType CPLGetValueType ( const char \* *pszValue* )**

Detect the type of the value contained in a string, whether it is a real, an integer or a string. Leading and trailing spaces are skipped in the analysis.

Note: in the context of this function, integer must be understood in a broad sense. It does not mean that the value can fit into a 32 bit integer for example. It might be larger.

**Parameters**

<i>pszValue</i>	the string to analyze
-----------------	-----------------------

**Returns**

returns the type of the value contained in the string.

**13.10.2.6 GByte\* CPLHexToBinary ( const char \* *pszHex*, int \* *pnBytes* )**

Hexadecimal to binary translation

**Parameters**

<i>pszHex</i>	the input hex encoded string.
<i>pnBytes</i>	the returned count of decoded bytes placed here.

**Returns**

returns binary buffer of data - free with CPLFree().

References CPLMalloc().

**13.10.2.7 int CPLIsUTF8 ( const char \* *pabyData*, int *nLen* )**

Test if a string is encoded as UTF-8.

**Parameters**

<i>pabyData</i>	input string to test
<i>nLen</i>	length of the input string, or -1 if the function must compute the string length. In which case it must be null terminated.

**Returns**

TRUE if the string is encoded as UTF-8. FALSE otherwise

**Since**

GDAL 1.7.0

**13.10.2.8 const char\* CPLParseNameValue ( const char \* pszNameValue, char \*\* ppszKey )**

Parse NAME=VALUE string into name and value components.

Note that if ppszKey is non-NULL, the key (or name) portion will be allocated using VSIMalloc(), and returned in that pointer. It is the applications responsibility to free this string, but the application should not modify or free the returned value portion.

This function also support "NAME:VALUE" strings and will strip white space from around the delimiter when forming name and value strings.

Eventually CSLFetchNameValue() and friends may be modified to use **CPLParseNameValue()** (p. ??).

**Parameters**

<i>pszNameValue</i>	string in "NAME=VALUE" format.
<i>ppszKey</i>	optional pointer though which to return the name portion.

**Returns**

the value portion (pointing into original string).

References CPLMalloc().

Referenced by CSLMerge(), and CSLSetNameValueSeparator().

**13.10.2.9 char\* CPLRecode ( const char \* pszSource, const char \* pszSrcEncoding, const char \* pszDstEncoding )**

Convert a string from a source encoding to a destination encoding.

The only guaranteed supported encodings are CPL\_ENC\_UTF8, CPL\_ENC\_ASCII and CPL\_ENC\_ISO8859\_1. Currently, the following conversions are supported :

- CPL\_ENC\_ASCII -> CPL\_ENC\_UTF8 or CPL\_ENC\_ISO8859\_1 (no conversion in fact)
- CPL\_ENC\_ISO8859\_1 -> CPL\_ENC\_UTF8
- CPL\_ENC\_UTF8 -> CPL\_ENC\_ISO8859\_1

If an error occurs an error may, or may not be posted with **CPLError()** (p. ??).

**Parameters**

<i>pszSource</i>	a NULL terminated string.
------------------	---------------------------

<i>pszSrcEncoding</i>	the source encoding.
<i>pszDstEncoding</i>	the destination encoding.

**Returns**

a NULL terminated string which should be freed with CPLFree().

**Since**

GDAL 1.6.0

References CPLStrdup().

**13.10.2.10** `char* CPLRecodeFromWChar ( const wchar_t * pwszSource, const char * pszSrcEncoding, const char * pszDstEncoding )`

Convert wchar\_t string to UTF-8.

Convert a wchar\_t string into a multibyte utf-8 string. The only guaranteed supported source encoding is CPL\_ENC\_UCS2, and the only guaranteed supported destination encodings are CPL\_ENC\_UTF8, CPL\_ENC\_ASCII and CPL\_ENC\_ISO8859\_1. In some cases (ie. using iconv()) other encodings may also be supported.

Note that the wchar\_t type varies in size on different systems. On win32 it is normally 2 bytes, and on unix 4 bytes.

If an error occurs an error may, or may not be posted with **CPLError()** (p. ??).

**Parameters**

<i>pwszSource</i>	the source wchar_t string, terminated with a 0 wchar_t.
<i>pszSrcEncoding</i>	the source encoding, typically CPL_ENC_UCS2.
<i>pszDstEncoding</i>	the destination encoding, typically CPL_ENC_UTF8.

**Returns**

a zero terminated multi-byte string which should be freed with CPLFree(), or NULL if an error occurs.

**Since**

GDAL 1.6.0

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

**13.10.2.11** `wchar_t* CPLRecodeToWChar ( const char * pszSource, const char * pszSrcEncoding, const char * pszDstEncoding )`

Convert UTF-8 string to a wchar\_t string.

Convert a 8bit, multi-byte per character input string into a wide character (wchar\_t) string. The only guaranteed supported source encodings are CPL\_ENC\_UTF8, CPL\_ENC\_ASCII and CPL\_ENC\_ISO8869\_1 (LATIN1). The only guaranteed supported destination encoding is CPL\_ENC\_UCS2. Other source and destination encodings may be supported depending on the underlying implementation.

Note that the wchar\_t type varies in size on different systems. On win32 it is normally 2 bytes, and on unix 4 bytes.

If an error occurs an error may, or may not be posted with **CPLError()** (p. ??).

**Parameters**

<i>pszSource</i>	input multi-byte character string.
<i>pszSrcEncoding</i>	source encoding, typically CPL_ENC_UTF8.
<i>pszDstEncoding</i>	destination encoding, typically CPL_ENC_UCS2.

**Returns**

the zero terminated wchar\_t string (to be freed with CPLFree()) or NULL on error.

**Since**

GDAL 1.6.0

**13.10.2.12 size\_t CPLStrlcat ( char \* pszDest, const char \* pszSrc, size\_t nDestSize )**

Appends a source string to a destination buffer.

This function ensures that the destination buffer is always NUL terminated (provided that its length is at least 1 and that there is at least one byte free in pszDest, that is to say strlen(pszDest\_before) < nDestSize)

This function is designed to be a safer, more consistent, and less error prone replacement for strncat. Its contract is identical to libbsd's strlcat.

Truncation can be detected by testing if the return value of CPLStrlcat is greater or equal to nDestSize.

```
char szDest[5];
CPLStrlcpy(szDest, "ab", sizeof(szDest));
if (CPLStrlcat(szDest, "cde", sizeof(szDest)) >= sizeof(szDest))
    fprintf(stderr, "truncation occurred !\n");
```

**Parameters**

<i>pszDest</i>	destination buffer. Must be NUL terminated before running CPLStrlcat
<i>pszSrc</i>	source string. Must be NUL terminated
<i>nDestSize</i>	size of destination buffer (including space for the NUL terminator character)

**Returns**

the thoretical length of the destination string after concatenation (=strlen(pszDest\_before) + strlen(pszSrc)). If strlen(pszDest\_before) >= nDestSize, then it returns nDestSize + strlen(pszSrc)

**Since**

GDAL 1.7.0

References CPLStrlcpy().

Referenced by CPLFormFilename(), CPLProjectRelativeFilename(), and CPLResetExtension().

**13.10.2.13 size\_t CPLStrlcpy ( char \* pszDest, const char \* pszSrc, size\_t nDestSize )**

Copy source string to a destination buffer.

This function ensures that the destination buffer is always NUL terminated (provided that its length is at least 1).

This function is designed to be a safer, more consistent, and less error prone replacement for strncpy. Its contract is identical to libbsd's strlcpy.

Truncation can be detected by testing if the return value of CPLStrlcpy is greater or equal to nDestSize.

```
char szDest[5];
if (CPLStrlcpy(szDest, "abcde", sizeof(szDest)) >= sizeof(szDest))
    fprintf(stderr, "truncation occurred !\n");
```



## Parameters

<i>pszDest</i>	destination buffer
<i>pszSrc</i>	source string. Must be NUL terminated
<i>nDestSize</i>	size of destination buffer (including space for the NUL terminator character)

## Returns

the length of the source string (=strlen(pszSrc))

## Since

GDAL 1.7.0

Referenced by CPLCleanTrailingSlash(), CPLFormFilename(), CPLGetBasename(), CPLGetDirname(), CPLGetExtension(), CPLGetPath(), CPLProjectRelativeFilename(), CPLResetExtension(), and CPLStrcat().

13.10.2.14 size\_t CPLStrnlen ( const char \* *pszStr*, size\_t *nMaxLen* )

Returns the length of a NUL terminated string by reading at most the specified number of bytes.

The **CPLStrnlen()** (p. ??) function returns MIN(strlen(pszStr), nMaxLen). Only the first nMaxLen bytes of the string will be read. Usefull to test if a string contains at least nMaxLen characters without reading the full string up to the NUL terminating character.

## Parameters

<i>pszStr</i>	a NUL terminated string
<i>nMaxLen</i>	maximum number of bytes to read in pszStr

## Returns

strlen(pszStr) if the length is lesser than nMaxLen, otherwise nMaxLen if the NUL character has not been found in the first nMaxLen bytes.

## Since

GDAL 1.7.0

Referenced by OGRSpatialReference::importFromPCI().

13.10.2.15 char\* CPLUnescapeString ( const char \* *pszInput*, int \* *pnLength*, int *nScheme* )

Unescape a string.

This function does the opposite of **CPLEscapeString()** (p. ??). Given a string with special values escaped according to some scheme, it will return a new copy of the string returned to it's original form.

## Parameters

<i>pszInput</i>	the input string. This is a zero terminated string.
<i>pnLength</i>	location to return the length of the unescaped string, which may in some cases include embedded '\0' characters.

<i>nScheme</i>	the escaped scheme to undo (see <b>CPLEscapeString()</b> (p. ??) for a list).
----------------	---

**Returns**

a copy of the unescaped string that should be freed by the application using CPLFree() when no longer needed.

References CPLDebug(), CPLMalloc(), and CPLRecodeFromWChar().

### 13.10.2.16 CPLString CPLURLAddKVP ( const char \* *pszURL*, const char \* *pszKey*, const char \* *pszValue* )

Return a new URL with a new key=value pair.

**Parameters**

<i>pszURL</i>	the URL.
<i>pszKey</i>	the key to find.
<i>pszValue</i>	the value of the key (may be NULL to unset an existing KVP).

**Returns**

the modified URL.

**Since**

GDAL 1.9.0

References CPLString::ifind().

### 13.10.2.17 CPLString CPLURLGetValue ( const char \* *pszURL*, const char \* *pszKey* )

Return the value matching a key from a key=value pair in a URL.

**Parameters**

<i>pszURL</i>	the URL.
<i>pszKey</i>	the key to find.

**Returns**

the value of empty string if not found.

**Since**

GDAL 1.9.0

References CPLString::ifind().

### 13.10.2.18 int CSLCount ( char \*\* *papszStrList* )

Return number of items in a string list.

Returns the number of items in a string list, not counting the terminating NULL. Passing in NULL is safe, and will result in a count of zero.

Lists are counted by iterating through them so long lists will take more time than short lists. Care should be taken to avoid using **CSLCount()** (p. ??) as an end condition for loops as it will result in  $O(n^2)$  behavior.

## Parameters

<i>papszStrList</i>	the string list to count.
---------------------	---------------------------

## Returns

the number of entries.

Referenced by OGRSFDriverRegistrar::AutoLoadDrivers(), CPLStringList::Count(), CPLCorrespondingPaths(), CPLHTTPParseMultipartMime(), CSLDuplicate(), CSLSetNameValueSeparator(), OGRFeature::Equal(), OGRDataSource::ExecuteSQL(), OGRSpatialReference::exportToPCI(), OGRSpatialReference::GetAttrNode(), OGRStyleTable::GetStyleName(), OGRSpatialReference::importFromOzi(), OGRSpatialReference::importFromPCI(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::importFromWMSAUTO(), OGRStyleTable::IsExist(), OGRFeature::SetField(), OGRSpatialReference::SetNode(), and VSIReadDirRecursive().

### 13.10.2.19 void CSLDestroy ( char \*\* *papszStrList* )

Free string list.

Frees the passed string list (null terminated array of strings). It is safe to pass NULL.

## Parameters

<i>papszStrList</i>	the list to free.
---------------------	-------------------

Referenced by OGRSFDriverRegistrar::AutoLoadDrivers(), OGRStyleTable::Clear(), CPLODBCStatement::Clear(), CPLStringList::Clear(), CPLGetXMLNode(), CPLHTTPDestroyResult(), CPLHTTPParseMultipartMime(), CPLSetThreadLocalConfigOption(), CPLSetXMLValue(), CPLUnlinkTree(), OGRDataSource::ExecuteSQL(), OGRSpatialReference::exportToPCI(), OGR\_SRSNode::exportToWkt(), OGRSpatialReference::GetAttrNode(), OGRStyleMgr::GetPart(), OGRSpatialReference::importFromOzi(), OGRSpatialReference::importFromPCI(), OGRSpatialReference::importFromProj4(), OGRSpatialReference::importFromWMSAUTO(), OGRStyleTable::LoadStyleTable(), OGRSFDriverRegistrar::RegisterDriver(), OGRFeature::SetField(), OGRUnionLayer::SetIgnoredFields(), OGRSpatialReference::SetNode(), OGRFeature::UnsetField(), and VSIReadDirRecursive().

### 13.10.2.20 char\*\* CSLDuplicate ( char \*\* *papszStrList* )

Clone a string list.

Efficiently allocates a copy of a string list. The returned list is owned by the caller and should be freed with **CSLDestroy()** (p. ??).

## Parameters

<i>papszStrList</i>	the input string list.
---------------------	------------------------

## Returns

newly allocated copy.

References CPLMalloc(), CPLStrdup(), and CSLCount().

Referenced by OGRStyleTable::Clone(), CSLMerge(), OGRFeature::SetField(), and OGRUnionLayer::SetIgnoredFields().

### 13.10.2.21 int CSLFindName ( char \*\* *papszStrList*, const char \* *pszName* )

Find StringList entry with given key name.

## Parameters

<i>papszStrList</i>	the string list to search.
<i>pszName</i>	the key value to look for (case insensitive).

## Returns

-1 on failure or the list index of the first occurrence matching the given key.

Referenced by CPLStringList::FindName().

### 13.10.2.22 int CSLFindString ( char \*\* *papszList*, const char \* *pszTarget* )

Find a string within a string list.

Returns the index of the entry in the string list that contains the target string. The string in the string list must be a full match for the target, but the search is case insensitive.

## Parameters

<i>papszList</i>	the string list to be searched.
<i>pszTarget</i>	the string to be searched for.

## Returns

the index of the string within the list or -1 on failure.

Referenced by OGR\_SRSNode::FixupOrdering().

### 13.10.2.23 char\*\* CSLLoad ( const char \* *pszFname* )

Load a text file into a string list.

The VSI\*L API is used, so **VSIFOpenL()** (p. ??) supported objects that aren't physical files can also be accessed. Files are returned as a string list, with one item in the string list per line. End of line markers are stripped (by **CPLReadLineL()** (p. ??)).

If reading the file fails a **CPLERROR()** (p. ??) will be issued and NULL returned.

## Parameters

<i>pszFname</i>	the name of the file to read.
-----------------	-------------------------------

## Returns

a string list with the files lines, now owned by caller. To be freed with **CSLDestroy()** (p. ??)

References CSLLoad2().

Referenced by OGRStyleTable::LoadStyleTable().

### 13.10.2.24 char\*\* CSLLoad2 ( const char \* *pszFname*, int *nMaxLines*, int *nMaxCols*, char \*\* *papszOptions* )

Load a text file into a string list.

The VSI\*L API is used, so **VSIFOpenL()** (p. ??) supported objects that aren't physical files can also be accessed. Files are returned as a string list, with one item in the string list per line. End of line markers are stripped (by **CPLReadLineL()** (p. ??)).

If reading the file fails a **CPLERROR()** (p. ??) will be issued and NULL returned.

## Parameters

<i>pszFname</i>	the name of the file to read.
<i>nMaxLines</i>	maximum number of lines to read before stopping, or -1 for no limit.
<i>nMaxCols</i>	maximum number of characters in a line before stopping, or -1 for no limit.
<i>papszOptions</i>	NULL-terminated array of options. Unused for now.

## Returns

a string list with the files lines, now owned by caller. To be freed with **CSLDestroy()** (p. ??)

## Since

GDAL 1.7.0

References CPLError(), CPLErrorReset(), CPLReadLine2L(), CPLReadLineL(), CPLStrdup(), VSIFCloseL(), VSIF-EofL(), and VSIFOpenL().

Referenced by CSLLoad().

#### 13.10.2.25 char\*\* CSLMerge ( char \*\* *papszOrig*, char \*\* *papszOverride* )

Merge two lists.

The two lists are merged, ensuring that if any keys appear in both that the value from the second (*papszOverride*) list take precedence.

## Parameters

<i>papszOrig</i>	the original list, being modified.
<i>papszOverride</i>	the list of items being merged in. This list is unaltered and remains owned by the caller.

## Returns

updated list.

References CPLParseNameValue(), CSLDuplicate(), and CSLSetNameValue().

#### 13.10.2.26 int CSLPartialFindString ( char \*\* *papszHaystack*, const char \* *pszNeedle* )

Find a substring within a string list.

Returns the index of the entry in the string list that contains the target string as a substring. The search is case sensitive (unlike **CSLFindString()** (p. ??)).

## Parameters

<i>papszHaystack</i>	the string list to be searched.
<i>pszNeedle</i>	the substring to be searched for.

## Returns

the index of the string within the list or -1 on failure.

#### 13.10.2.27 char\*\* CSLSetNameValue ( char \*\* *papszList*, const char \* *pszName*, const char \* *pszValue* )

Assign value to name in StringList.

Set the value for a given name in a `StringList` of "Name=Value" pairs ("Name:Value" pairs are also supported for backward compatibility with older stuff.)

If there is already a value for that name in the list then the value is changed, otherwise a new "Name=Value" pair is added.

## Parameters

<i>papszList</i>	the original list, the modified version is returned.
<i>pszName</i>	the name to be assigned a value. This should be a well formed token (no spaces or very special characters).
<i>pszValue</i>	the value to assign to the name. This should not contain any newlines (CR or LF) but is otherwise pretty much unconstrained. If NULL any corresponding value will be removed.

## Returns

modified stringlist.

References CPLMalloc().

Referenced by CPLSetConfigOption(), CPLSetThreadLocalConfigOption(), and CSLMerge().

### 13.10.2.28 void CSLSetNameValueSeparator ( char \*\* *papszList*, const char \* *pszSeparator* )

Replace the default separator (":" or "=") with the passed separator in the given name/value list.

Note that if a separator other than ":" or "=" is used, the resulting list will not be manipulatable by the CSL name/value functions any more.

The **CPLParseNameValue()** (p. ??) function is used to break the existing lines, and it also strips white space from around the existing delimiter, thus the old separator, and any white space will be replaced by the new separator. For formatting purposes it may be desirable to include some white space in the new separator. eg. ": " or "= ".

## Parameters

<i>papszList</i>	the list to update. Component strings may be freed but the list array will remain at the same location.
<i>pszSeparator</i>	the new separator string to insert.

References CPLMalloc(), CPLParseNameValue(), and CSLCount().

### 13.10.2.29 int CSLTestBoolean ( const char \* *pszValue* )

Test what boolean value contained in the string.

If pszValue is "NO", "FALSE", "OFF" or "0" will be returned FALSE. Otherwise, TRUE will be returned.

## Parameters

<i>pszValue</i>	the string should be tested.
-----------------	------------------------------

## Returns

TRUE or FALSE.

Referenced by OGRLayer::Clip(), CPLHTTPFetch(), OGRGeometry::dumpReadable(), OGRFeature::DumpReadable(), OGRLayer::Erase(), OGRSpatialReference::exportToProj4(), CPLStringList::FetchBoolean(), OGRLayer::Identity(), OGRLayer::Intersection(), OGR\_G\_CreateFromGML(), OGR\_G\_ExportToGMLEx(), OGRGeocodeCreateSession(), OGRGeometryFactory::organizePolygons(), OGRFeature::SetField(), OGRLayer::SymDifference(), OGRLineString::transform(), OGRLayer::Union(), and OGRLayer::Update().

### 13.10.2.30 char\*\* CSLTokenizeString2 ( const char \* *pszString*, const char \* *pszDelimiters*, int *nCSLTFlags* )

Tokenize a string.

This function will split a string into tokens based on specified' delimiter(s) with a variety of options. The returned result is a string list that should be freed with **CSLDestroy()** (p. ??) when no longer needed.

The available parsing options are:

- **CSLT\_ALLOWEMPTYTOKENS**: allow the return of empty tokens when two delimiters in a row occur with no other text between them. If not set, empty tokens will be discarded;
- **CSLT\_STRIPLEADSPACES**: strip leading space characters from the token (as reported by `isspace()`);
- **CSLT\_STRIPENDSPACES**: strip ending space characters from the token (as reported by `isspace()`);
- **CSLT\_HONOURSTRINGS**: double quotes can be used to hold values that should not be broken into multiple tokens;
- **CSLT\_PRESERVEQUOTES**: string quotes are carried into the tokens when this is set, otherwise they are removed;
- **CSLT\_PRESERVEESCAPES**: if set backslash escapes (for backslash itself, and for literal double quotes) will be preserved in the tokens, otherwise the backslashes will be removed in processing.

### Example:

Parse a string into tokens based on various white space (space, newline, tab) and then print out results and cleanup. Quotes may be used to hold white space in tokens.

```
char **papszTokens;
int i;

papszTokens =
    CSLTokenizeString2( pszCommand, " \t\n",
                       CSLT_HONOURSTRINGS | CSLT_ALLOWEMPTYTOKENS );

for( i = 0; papszTokens != NULL && papszTokens[i] != NULL; i++ )
    printf( "arg %d: '%s'", papszTokens[i] );
CSLDestroy( papszTokens );
```

### Parameters

<i>pszString</i>	the string to be split into tokens.
<i>pszDelimiters</i>	one or more characters to be used as token delimiters.
<i>nCSLTFlags</i>	an ORing of one or more of the CSLT_ flag values.

### Returns

a string list of tokens owned by the caller.

References `CPLStringList::AddString()`, `CPLStringList::Assign()`, `CPLStringList::Count()`, `CPLCalloc()`, `CPLRealloc()`, and `CPLStringList::StealList()`.

Referenced by `OGRStyleMgr::GetPart()`, `OGRSpatialReference::importFromOzi()`, and `OGRFeature::SetField()`.

## 13.11 cpl\_vsi.h File Reference

```
#include "cpl_port.h"
#include <unistd.h>
#include <sys/stat.h>
```

### Functions

- **VSILFILE \*** **VSIFOpenL** (const char \*, const char \*)  
*Open file.*
- int **VSIFCloseL** (VSILFILE \*)



- Close file.*

  - int **VSIFSeekL** (VSILFILE \*, vsi\_l\_offset, int)

*Seek to requested offset.*
- vsi\_l\_offset **VSIFTellL** (VSILFILE \*)

*Tell current file offset.*
- size\_t **VSIFReadL** (void \*, size\_t, size\_t, VSILFILE \*)

*Read bytes from file.*
- int **VSIFReadMultiRangeL** (int nRanges, void \*\*ppData, const vsi\_l\_offset \*panOffsets, const size\_t \*panSizes, VSILFILE \*)

*Read several ranges of bytes from file.*
- size\_t **VSIFWriteL** (const void \*, size\_t, size\_t, VSILFILE \*)

*Write bytes to file.*
- int **VSIFEofL** (VSILFILE \*)

*Test for end of file.*
- int **VSIFTruncateL** (VSILFILE \*, vsi\_l\_offset)

*Truncate/expand the file to the specified size.*
- int **VSIFFlushL** (VSILFILE \*)

*Flush pending writes to disk.*
- int **VSIFPrintfL** (VSILFILE \*, const char \*,...)

*Formatted write to file.*
- int **VSISatL** (const char \*, VSISatBufL \*)

*Get filesystem object info.*
- int **VSISatExL** (const char \*pszFilename, VSISatBufL \*psStatBuf, int nFlags)

*Get filesystem object info.*
- int **VSILsCaseSensitiveFS** (const char \*pszFilename)

*Returns if the filenames of the filesystem are case sensitive.*
- void \* **VSIMalloc2** (size\_t nSize1, size\_t nSize2)
- void \* **VSIMalloc3** (size\_t nSize1, size\_t nSize2, size\_t nSize3)
- char \*\* **VSIReadDir** (const char \*)

*Read names in a directory.*
- char \*\* **VSIReadDirRecursive** (const char \*pszPath)

*Read names in a directory recursively.*
- int **VSIMkdir** (const char \*pathname, long mode)

*Create a directory.*
- int **VSIRmdir** (const char \*pathname)

*Delete a directory.*
- int **VSILUnlink** (const char \*pathname)

*Delete a file.*
- int **VSIRename** (const char \*oldpath, const char \*newpath)

*Rename a file.*
- void **VSILInstallMemFileHandler** (void)

*Install "memory" file system handler.*
- void **VSILInstallSubFileHandler** (void)
- void **VSILInstallCurlFileHandler** (void)

*Install /vsicurl/ HTTP/FTP file system handler (requires libcurl)*
- void **VSILInstallCurlStreamingFileHandler** (void)

*Install /vsicurl\_streaming/ HTTP/FTP file system handler (requires libcurl)*
- void **VSILInstallGZipFileHandler** (void)

*Install GZip file system handler.*
- void **VSILInstallZipFileHandler** (void)

*Install ZIP file system handler.*

- void **VSIInstallStdinHandler** (void)  
*Install /vsistdin/ file system handler.*
- void **VSIInstallStdoutHandler** (void)  
*Install /vsistdout/ file system handler.*
- void **VSIInstallSparseFileHandler** (void)
- void **VSIInstallTarFileHandler** (void)  
*Install /vsitar/ file system handler.*
- VSILFILE \* **VSIFileFromMemBuffer** (const char \*pszFilename, GByte \*pabyData, vsi\_l\_offset nDataLength, int bTakeOwnership)  
*Create memory "file" from a buffer.*
- GByte \* **VSIGetMemFileBuffer** (const char \*pszFilename, vsi\_l\_offset \*pnDataLength, int bUnlinkAndSeize)  
*Fetch buffer underlying memory file.*

### 13.11.1 Detailed Description

#### Standard C Covers

The VSI functions are intended to be hookable aliases for Standard C I/O, memory allocation and other system functions. They are intended to allow virtualization of disk I/O so that non file data sources can be made to appear as files, and so that additional error trapping and reporting can be interested. The memory access API is aliased so that special application memory management services can be used.

Is is intended that each of these functions retains exactly the same calling pattern as the original Standard C functions they relate to. This means we don't have to provide custom documentation, and also means that the default implementation is very simple.

### 13.11.2 Function Documentation

#### 13.11.2.1 int VSIFCloseL ( VSILFILE \* fp )

Close file.

This function closes the indicated file.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fclose() function.

#### Parameters

<i>fp</i>	file handle opened with <b>VSIFileOpenL()</b> (p. ??).
-----------	--

#### Returns

0 on success or -1 on failure.

Referenced by CPLCloseShared(), CPLParseXMLFile(), CPLSerializeXMLTreeToFile(), and CSLLoad2().

#### 13.11.2.2 int VSIFEOF ( VSILFILE \* fp )

Test for end of file.

Returns TRUE (non-zero) if an end-of-file condition occurred during the previous read operation. The end-of-file flag is cleared by a successful **VSISeekL()** (p. ??) call.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX feof() call.

## Parameters

<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??).
-----------	---

## Returns

TRUE if at EOF else FALSE.

Referenced by CSLoad2().

13.11.2.3 int VSIFlushL ( VSILFILE \* *fp* )

Flush pending writes to disk.

For files in write or update mode and on filesystem types where it is applicable, all pending output on the file is flushed to the physical disk.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fflush() call.

## Parameters

<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??).
-----------	---

## Returns

0 on success or -1 on error.

13.11.2.4 VSILFILE\* VSIFileFromMemBuffer ( const char \* *pszFilename*, GByte \* *pabyData*, vsi\_l\_offset *nDataLength*, int *bTakeOwnership* )

Create memory "file" from a buffer.

A virtual memory file is created from the passed buffer with the indicated filename. Under normal conditions the filename would need to be absolute and within the /vsimem/ portion of the filesystem.

If bTakeOwnership is TRUE, then the memory file system handler will take ownership of the buffer, freeing it when the file is deleted. Otherwise it remains the responsibility of the caller, but should not be freed as long as it might be accessed as a file. In no circumstances does this function take a copy of the pabyData contents.

## Parameters

<i>pszFilename</i>	the filename to be created.
<i>pabyData</i>	the data buffer for the file.
<i>nDataLength</i>	the length of buffer in bytes.
<i>bTakeOwnership</i>	TRUE to transfer "ownership" of buffer or FALSE.

## Returns

open file handle on created file (see **VSIFOpenL()** (p. ??)).

References VSIIInstallMemFileHandler().

13.11.2.5 VSILFILE\* VSIFOpenL ( const char \* *pszFilename*, const char \* *pszAccess* )

Open file.

This function opens a file with the desired access. Large files (larger than 2GB) should be supported. Binary access is always implied and the "b" does not need to be included in the pszAccess string.

Note that the "VSILFILE \*" returned since GDAL 1.8.0 by this function is *NOT* a standard C library FILE \*, and cannot be used with any functions other than the "VSI\*L" family of functions. They aren't "real" FILE objects.

On windows it is possible to define the configuration option GDAL\_FILE\_IS\_UTF8 to have pszFilename treated as being in the local encoding instead of UTF-8, retoring the pre-1.8.0 behavior of **VSIFOpenL()** (p. ??).

This method goes through the VSIFHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fopen() function.

#### Parameters

<i>pszFilename</i>	the file to open. UTF-8 encoded.
<i>pszAccess</i>	access requested (ie. "r", "r+", "w").

#### Returns

NULL on failure, or the file handle.

Referenced by CPLOpenShared(), CPLParseXMLFile(), CPLSerializeXMLTreeToFile(), and CSLLoad2().

#### 13.11.2.6 int VSIFPrintfL ( VSILFILE \* fp, const char \* pszFormat, ... )

Formatted write to file.

Provides fprintf() style formatted output to a VSI\*L file. This formats an internal buffer which is written using **VSIFWriteL()** (p. ??).

Analog of the POSIX fprintf() call.

#### Parameters

<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??).
<i>pszFormat</i>	the printf style format string.

#### Returns

the number of bytes written or -1 on an error.

References VSIFWriteL().

#### 13.11.2.7 size\_t VSIFReadL ( void \* pBuffer, size\_t nSize, size\_t nCount, VSILFILE \* fp )

Read bytes from file.

Reads nCount objects of nSize bytes from the indicated file at the current offset into the indicated buffer.

This method goes through the VSIFHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fread() call.

#### Parameters

<i>pBuffer</i>	the buffer into which the data should be read (at least nCount * nSize bytes in size).
<i>nSize</i>	size of objects to read in bytes.
<i>nCount</i>	number of objects to read.

<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??).
-----------	---

**Returns**

number of objects successfully read.

Referenced by CPLParseXMLFile(), and CPLReadLine2L().

**13.11.2.8** `int VSIFReadMultiRangeL ( int nRanges, void ** ppData, const vsi_l_offset * panOffsets, const size_t * panSizes, VSILFILE * fp )`

Read several ranges of bytes from file.

Reads *nRanges* objects of *panSizes*[*i*] bytes from the indicated file at the offset *panOffsets*[*i*] into the buffer *ppData*[*i*].

Ranges must be sorted in ascending start offset, and must not overlap each other.

This method goes through the VSIFFileHandler virtualization and may work on unusual filesystems such as in memory or /vsicurl/.

**Parameters**

<i>nRanges</i>	number of ranges to read.
<i>ppData</i>	array of <i>nRanges</i> buffer into which the data should be read ( <i>ppData</i> [ <i>i</i> ] must be at list <i>panSizes</i> [ <i>i</i> ] bytes).
<i>panOffsets</i>	array of <i>nRanges</i> offsets at which the data should be read.
<i>panSizes</i>	array of <i>nRanges</i> sizes of objects to read (in bytes).
<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??).

**Returns**

0 in case of success, -1 otherwise.

**Since**

GDAL 1.9.0

**13.11.2.9** `int VSIFSeekL ( VSILFILE * fp, vsi_l_offset nOffset, int nWhence )`

Seek to requested offset.

Seek to the desired offset (*nOffset*) in the indicated file.

This method goes through the VSIFFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX `fseek()` call.

**Parameters**

<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??).
<i>nOffset</i>	offset in bytes.
<i>nWhence</i>	one of SEEK_SET, SEEK_CUR or SEEK_END.

**Returns**

0 on success or -1 one failure.

Referenced by CPLParseXMLFile(), and CPLReadLine2L().

### 13.11.2.10 vsi\_l\_offset VSIFTell ( VSILFILE \* *fp* )

Tell current file offset.

Returns the current file read/write offset in bytes from the beginning of the file.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX ftell() call.

#### Parameters

<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??).
-----------	---

#### Returns

file offset in bytes.

Referenced by CPLParseXMLFile(), and CPLReadLine2L().

### 13.11.2.11 int VSIFTruncateL ( VSILFILE \* *fp*, vsi\_l\_offset *nNewSize* )

Truncate/expand the file to the specified size.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX ftruncate() call.

#### Parameters

<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??).
<i>nNewSize</i>	new size in bytes.

#### Returns

0 on success

#### Since

GDAL 1.9.0

### 13.11.2.12 size\_t VSIFWriteL ( const void \* *pBuffer*, size\_t *nSize*, size\_t *nCount*, VSILFILE \* *fp* )

Write bytes to file.

Writes *nCount* objects of *nSize* bytes to the indicated file at the current offset into the indicated buffer.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX fwrite() call.

#### Parameters

<i>pBuffer</i>	the buffer from which the data should be written (at least <i>nCount</i> * <i>nSize</i> bytes in size).
<i>nSize</i>	size of objects to read in bytes.

<i>nCount</i>	number of objects to read.
<i>fp</i>	file handle opened with <b>VSIFOpenL()</b> (p. ??).

**Returns**

number of objects successfully written.

Referenced by CPLSerializeXMLTreeToFile(), and VSIFPrintfL().

### 13.11.2.13 GByte\* VSIGetMemFileBuffer ( const char \* *pszFilename*, vsi\_l\_offset \* *pnDataLength*, int *bUnlinkAndSeize* )

Fetch buffer underlying memory file.

This function returns a pointer to the memory buffer underlying a virtual "in memory" file. If *bUnlinkAndSeize* is TRUE the filesystem object will be deleted, and ownership of the buffer will pass to the caller otherwise the underlying file will remain in existence.

**Parameters**

<i>pszFilename</i>	the name of the file to grab the buffer of.
<i>pnDataLength</i>	(file) length returned in this variable.
<i>bUnlinkAndSeize</i>	TRUE to remove the file, or FALSE to leave unaltered.

**Returns**

pointer to memory buffer or NULL on failure.

References CPLDebug().

### 13.11.2.14 void VSInstallCurlFileHandler ( void )

Install /vsicurl/ HTTP/FTP file system handler (requires libcurl)

A special file handler is installed that allows reading on-the-fly of files available through HTTP/FTP web protocols, without downloading the entire file.

Recognized filenames are of the form /vsicurl/http://path/to/remote/ressource or /vsicurl/ftp://path/to/remote/ressource where path/to/remote/ressource is the URL of a remote ressource.

Partial downloads (requires the HTTP server to support random reading) are done with a 16 KB granularity by default. If the driver detects sequential reading it will progressively increase the chunk size up to 2 MB to improve download performance.

The GDAL\_HTTP\_PROXY, GDAL\_HTTP\_PROXYUSERPWD and GDAL\_PROXY\_AUTH configuration options can be used to define a proxy server. The syntax to use is the one of Curl CURLOPT\_PROXY, CURLOPT\_PROXYUSERPWD and CURLOPT\_PROXYAUTH options.

Starting with GDAL 1.10, the file can be cached in RAM by setting the configuration option VSI\_CACHE to TRUE. The cache size defaults to 25 MB, but can be modified by setting the configuration option VSI\_CACHE\_SIZE (in bytes).

**VSIStatL()** (p. ??) will return the size in *st\_size* member and file nature- file or directory - in *st\_mode* member (the later only reliable with FTP resources for now).

**VSIReadDir()** (p. ??) should be able to parse the HTML directory listing returned by the most popular web servers, such as Apache or Microsoft IIS.

This special file handler can be combined with other virtual filesystems handlers, such as /vsizip. For example, /vsizip/vsicurl/path/to/remote/file.zip/path/inside/zip

Since

GDAL 1.8.0

### 13.11.2.15 void VSInstallCurlStreamingFileHandler ( void )

Install /vsicurl\_streaming/ HTTP/FTP file system handler (requires libcurl)

A special file handler is installed that allows on-the-fly reading of files streamed through HTTP/FTP web protocols (typically dynamically generated files), without downloading the entire file.

Although this file handler is able seek to random offsets in the file, this will not be efficient. If you need efficient random access and that the server supports range downloading, you should use the /vsicurl/ file system handler instead.

Recognized filenames are of the form /vsicurl\_streaming/http://path/to/remote/ressource or /vsicurl\_streaming/ftp://path/to/remote/ressource where path/to/remote/ressource is the URL of a remote ressource.

The GDAL\_HTTP\_PROXY, GDAL\_HTTP\_PROXYUSERPWD and GDAL\_PROXY\_AUTH configuration options can be used to define a proxy server. The syntax to use is the one of Curl CURLOPT\_PROXY, CURLOPT\_PROXYUSERPWD and CURLOPT\_PROXYAUTH options.

The file can be cached in RAM by setting the configuration option VSI\_CACHE to TRUE. The cache size defaults to 25 MB, but can be modified by setting the configuration option VSI\_CACHE\_SIZE (in bytes).

**VSIStatL()** (p. ??) will return the size in st\_size member and file nature- file or directory - in st\_mode member (the later only reliable with FTP resources for now).

Since

GDAL 1.10

### 13.11.2.16 void VSInstallGZipFileHandler ( void )

Install GZip file system handler.

A special file handler is installed that allows reading on-the-fly and writing in GZip (.gz) files.

All portions of the file system underneath the base path "/vsigzip/" will be handled by this driver.

Additional documentation is to be found at <http://trac.osgeo.org/gdal/wiki/UserDocs/ReadInZip>

Since

GDAL 1.6.0

### 13.11.2.17 void VSInstallMemFileHandler ( void )

Install "memory" file system handler.

A special file handler is installed that allows block of memory to be treated as files. All portions of the file system underneath the base path "/vsimem/" will be handled by this driver.

Normal VSI\*L functions can be used freely to create and destroy memory arrays treating them as if they were real file system objects. Some additional methods exist to efficient create memory file system objects without duplicating original copies of the data or to "steal" the block of memory associated with a memory file.

At this time the memory handler does not properly handle directory semantics for the memory portion of the filesystem. The **VSIReadDir()** (p. ??) function is not supported though this will be corrected in the future.

Calling this function repeatedly should do no harm, though it is not necessary. It is already called the first time a virtualizable file access function (ie. **VSIOpenL()** (p. ??), **VSIMkdir()**, etc) is called.



This code example demonstrates using GDAL to translate from one memory buffer to another.

```
* GByte *ConvertBufferFormat( GByte *pabyInData, vsi_l_offset nInDataLength,
*                               vsi_l_offset *pnOutDataLength )
* {
*     // create memory file system object from buffer.
*     VSIFCloseL( VSIFileFromMemBuffer( "/vsimem/work.dat", pabyInData,
*                                       nInDataLength, FALSE ) );
*
*     // Open memory buffer for read.
*     GDALDatasetH hDS = GDALOpen( "/vsimem/work.dat", GA_ReadOnly );
*
*     // Get output format driver.
*     GDALDriverH hDriver = GDALGetDriverByName( "GTiff" );
*     GDALDatasetH hOutDS;
*
*     hOutDS = GDALCreateCopy( hDriver, "/vsimem/out.tif", hDS, TRUE, NULL,
*                             NULL, NULL );
*
*     // close source file, and "unlink" it.
*     GDALClose( hDS );
*     VSIUnlink( "/vsimem/work.dat" );
*
*     // seize the buffer associated with the output file.
*
*     return VSIGetMemFileBuffer( "/vsimem/out.tif", pnOutDataLength, TRUE );
* }
*
```

Referenced by VSIFileFromMemBuffer().

### 13.11.2.18 void VSInstallSparseFileHandler( void )

Install /vsisparse/ virtual file handler.

The sparse virtual file handler allows a virtual file to be composed from chunks of data in other files, potentially with large spaces in the virtual file set to a constant value. This can make it possible to test some sorts of operations on what seems to be a large file with image data set to a constant value. It is also helpful when wanting to add test files to the test suite that are too large, but for which most of the data can be ignored. It could, in theory, also be used to treat several files on different file systems as one large virtual file.

The file referenced by /vsisparse/ should be an XML control file formatted something like:

```
<VSISparseFile>
  <Length>87629264</Length>
  <SubfileRegion> Stuff at start of file.
    <Filename relative="1">251_head.dat</Filename>
    <DestinationOffset>0</DestinationOffset>
    <SourceOffset>0</SourceOffset>
    <RegionLength>2768</RegionLength>
  </SubfileRegion>

  <SubfileRegion> RasterDMS node.
    <Filename relative="1">251_rasterdms.dat</Filename>
    <DestinationOffset>87313104</DestinationOffset>
    <SourceOffset>0</SourceOffset>
    <RegionLength>160</RegionLength>
  </SubfileRegion>

  <SubfileRegion> Stuff at end of file.
    <Filename relative="1">251_tail.dat</Filename>
    <DestinationOffset>87611924</DestinationOffset>
    <SourceOffset>0</SourceOffset>
    <RegionLength>17340</RegionLength>
  </SubfileRegion>

  <ConstantRegion> Default for the rest of the file.
    <DestinationOffset>0</DestinationOffset>
    <RegionLength>87629264</RegionLength>
    <Value>0</Value>
  </ConstantRegion>
</VSISparseFile>
```

Hopefully the values and semantics are fairly obvious.

This driver is installed by default.

#### 13.11.2.19 void VSInstallStdinHandler ( void )

Install /vsistdin/ file system handler.

A special file handler is installed that allows reading from the standard input stream.

The file operations available are of course limited to Read() and forward Seek() (full seek in the first MB of a file).

Since

GDAL 1.8.0

#### 13.11.2.20 void VSInstallStdoutHandler ( void )

Install /vsistdout/ file system handler.

A special file handler is installed that allows writing to the standard output stream.

The file operations available are of course limited to Write().

Since

GDAL 1.8.0

#### 13.11.2.21 void VSInstallSubFileHandler ( void )

Install /vsisubfile/ virtual file handler.

This virtual file system handler allows access to subregions of files, treating them as a file on their own to the virtual file system functions (**VSIFOpenL()** (p. ??), etc).

A special form of the filename is used to indicate a subportion of another file:

/vsisubfile/<offset>[\_<size>],<filename>

The size parameter is optional. Without it the remainder of the file from the start offset as treated as part of the subfile. Otherwise only <size> bytes from <offset> are treated as part of the subfile. The <filename> portion may be a relative or absolute path using normal rules. The <offset> and <size> values are in bytes.

eg. /vsisubfile/1000\_3000,/data/abc.ntf /vsisubfile/5000,.../xyz/raw.dat

Unlike the /vsimem/ or conventional file system handlers, there is no meaningful support for filesystem operations for creating new files, traversing directories, and deleting files within the /vsisubfile/ area. Only the **VSISStatL()** (p. ??), **VSIFOpenL()** (p. ??) and operations based on the file handle returned by **VSIFOpenL()** (p. ??) operate properly.

#### 13.11.2.22 void VSInstallTarFileHandler ( void )

Install /vsitar/ file system handler.

A special file handler is installed that allows reading on-the-fly in TAR (regular .tar, or compressed .tar.gz/.tgz) archives.

All portions of the file system underneath the base path "/vsitar/" will be handled by this driver.

The syntax to open a file inside a zip file is /vsitar/path/to/the/file.tar/path/inside/the/tar/file were path/to/the/file.tar is relative or absolute and path/inside/the/tar/file is the relative path to the file inside the archive.

If the path is absolute, it should begin with a / on a Unix-like OS (or C:\ on Windows), so the line looks like /vsitar//home/gdal/... For example gdalinfo /vsitar/myarchive.tar/subdir1/file1.tif

Syntactic sugar : if the tar archive contains only one file located at its root, just mentionning `"/vsitar/path/to/the/file.tar"` will work

**VSIStatL()** (p. ??) will return the uncompressed size in `st_size` member and file nature- file or directory - in `st_mode` member.

Directory listing is available through **VSIReadDir()** (p. ??).

Since

GDAL 1.8.0

#### 13.11.2.23 void VSInstallZipFileHandler ( void )

Install ZIP file system handler.

A special file handler is installed that allows reading on-the-fly in ZIP (.zip) archives.

All portions of the file system underneath the base path `"/vsizip/"` will be handled by this driver.

The syntax to open a file inside a zip file is `/vsizip/path/to/the/file.zip/path/inside/the/zip/file` where `path/to/the/file.zip` is relative or absolute and `path/inside/the/zip/file` is the relative path to the file inside the archive.

If the path is absolute, it should begin with a `/` on a Unix-like OS (or `C:\` on Windows), so the line looks like `/vsizip/home/gdal/...` For example `gdalinfo /vsizip/myarchive.zip/subdir1/file1.tif`

Syntactic sugar : if the .zip file contains only one file located at its root, just mentionning `"/vsizip/path/to/the/file.zip"` will work

**VSIStatL()** (p. ??) will return the uncompressed size in `st_size` member and file nature- file or directory - in `st_mode` member.

Directory listing is available through **VSIReadDir()** (p. ??).

Since GDAL 1.8.0, write capabilities are available. They allow creating a new zip file and adding new files to an already existing (or just created) zip file. Read and write operations cannot be interleaved : the new zip must be closed before being re-opened for read.

Additional documentation is to be found at <http://trac.osgeo.org/gdal/wiki/UserDocs/ReadIn-Zip>

Since

GDAL 1.6.0

#### 13.11.2.24 int VSIsCaseSensitiveFS ( const char \* pszFilename )

Returns if the filenames of the filesystem are case sensitive.

This method retrieves to which filesystem belongs the passed filename and return TRUE if the filenames of that filesystem are case sensitive.

Currently, this will return FALSE only for Windows real filenames. Other VSI virtual filesystems are case sensitive.

This methods avoid ugly `#ifndef WIN32 / #endif` code, that is wrong when dealing with virtual filenames.

Parameters

<i>pszFilename</i>	the path of the filesystem object to be tested. UTF-8 encoded.
--------------------	--

Returns

TRUE if the filenames of the filesystem are case sensitive.

Since

GDAL 1.8.0

Referenced by CPLFormCIFilename().

**13.11.2.25** void\* VSIMalloc2 ( size\_t nSize1, size\_t nSize2 )

VSIMalloc2 allocates (nSize1 \* nSize2) bytes. In case of overflow of the multiplication, or if memory allocation fails, a NULL pointer is returned and a CE\_Failure error is raised with **CPLError()** (p. ??). If nSize1 == 0 || nSize2 == 0, a NULL pointer will also be returned. CPLFree() or VSIFree() can be used to free memory allocated by this function.

References CPLError().

Referenced by OGRPolygon::importFromWkb().

**13.11.2.26** void\* VSIMalloc3 ( size\_t nSize1, size\_t nSize2, size\_t nSize3 )

VSIMalloc3 allocates (nSize1 \* nSize2 \* nSize3) bytes. In case of overflow of the multiplication, or if memory allocation fails, a NULL pointer is returned and a CE\_Failure error is raised with **CPLError()** (p. ??). If nSize1 == 0 || nSize2 == 0 || nSize3 == 0, a NULL pointer will also be returned. CPLFree() or VSIFree() can be used to free memory allocated by this function.

References CPLError().

**13.11.2.27** int VSIMkdir ( const char \* pszPathname, long mode )

Create a directory.

Create a new directory with the indicated mode. The mode is ignored on some platforms. A reasonable default mode value would be 0666. This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX mkdir() function.

Parameters

<i>pszPathname</i>	the path to the directory to create. UTF-8 encoded.
<i>mode</i>	the permissions mode.

Returns

0 on success or -1 on an error.

**13.11.2.28** char\*\* VSIReadDir ( const char \* pszPath )

Read names in a directory.

This function abstracts access to directory contents. It returns a list of strings containing the names of files, and directories in this directory. The resulting string list becomes the responsibility of the application and should be freed with **CSLDestroy()** (p. ??) when no longer needed.

Note that no error is issued via **CPLError()** (p. ??) if the directory path is invalid, though NULL is returned.

This function used to be known as CPLReadDir(), but the old name is now deprecated.

## Parameters

<i>pszPath</i>	the relative, or absolute path of a directory to read. UTF-8 encoded.
----------------	---

## Returns

The list of entries in the directory, or NULL if the directory doesn't exist. Filenames are returned in UTF-8 encoding.

Referenced by VSIReadDirRecursive().

13.11.2.29 `char** VSIReadDirRecursive ( const char * pszPathIn )`

Read names in a directory recursively.

This function abstracts access to directory contents and subdirectories. It returns a list of strings containing the names of files and directories in this directory and all subdirectories. The resulting string list becomes the responsibility of the application and should be freed with **CSLDestroy()** (p. ??) when no longer needed.

Note that no error is issued via **CPLError()** (p. ??) if the directory path is invalid, though NULL is returned.

## Parameters

<i>pszPathIn</i>	the relative, or absolute path of a directory to read. UTF-8 encoded.
------------------	---

## Returns

The list of entries in the directory and subdirectories or NULL if the directory doesn't exist. Filenames are returned in UTF-8 encoding.

## Since

GDAL 1.10.0

References CPLStringList::AddString(), CPLStrdup(), CSLCount(), CSLDestroy(), CPLStringList::StealList(), VSIReadDir(), and VSISatL().

13.11.2.30 `int VSIRename ( const char * oldpath, const char * newpath )`

Rename a file.

Renames a file object in the file system. It should be possible to rename a file onto a new filesystem, but it is safest if this function is only used to rename files that remain in the same directory.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX rename() function.

## Parameters

<i>oldpath</i>	the name of the file to be renamed. UTF-8 encoded.
<i>newpath</i>	the name the file should be given. UTF-8 encoded.

## Returns

0 on success or -1 on an error.

### 13.11.2.31 int VSIRmdir ( const char \* *pszDirname* )

Delete a directory.

Deletes a directory object from the file system. On some systems the directory must be empty before it can be deleted.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX rmdir() function.

#### Parameters

<i>pszDirname</i>	the path of the directory to be deleted. UTF-8 encoded.
-------------------	---

#### Returns

0 on success or -1 on an error.

Referenced by CPLUnlinkTree().

### 13.11.2.32 int VSISatExL ( const char \* *pszFilename*, VSISatBufL \* *psStatBuf*, int *nFlags* )

Get filesystem object info.

Fetches status information about a filesystem object (file, directory, etc). The returned information is placed in the VSISatBufL structure. For portability only the st\_size (size in bytes), and st\_mode (file type). This method is similar to VSISat(), but will work on large files on systems where this requires special calls.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX stat() function, with an extra parameter to specify which information is needed, which offers a potential for speed optimizations on specialized and potentially slow virtual filesystem objects (/vsigzip/, /vsicurl/)

#### Parameters

<i>pszFilename</i>	the path of the filesystem object to be queried. UTF-8 encoded.
<i>psStatBuf</i>	the structure to load with information.
<i>nFlags</i>	0 to get all information, or VSI_STAT_EXISTS_FLAG, VSI_STAT_NATURE_FLAG or VSI_STAT_SIZE_FLAG, or a combination of those to get partial info.

#### Returns

0 on success or -1 on an error.

#### Since

GDAL 1.8.0

Referenced by CPLFormCIFilename(), and VSISatL().

### 13.11.2.33 int VSISatL ( const char \* *pszFilename*, VSISatBufL \* *psStatBuf* )

Get filesystem object info.

Fetches status information about a filesystem object (file, directory, etc). The returned information is placed in the VSISatBufL structure. For portability only the st\_size (size in bytes), and st\_mode (file type). This method is similar to VSISat(), but will work on large files on systems where this requires special calls.

This method goes through the VSIFileHandler virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX stat() function.

## Parameters

<i>pszFilename</i>	the path of the filesystem object to be queried. UTF-8 encoded.
<i>psStatBuf</i>	the structure to load with information.

## Returns

0 on success or -1 on an error.

References VSISatExL().

Referenced by OGRSFDriverRegistrar::AutoLoadDrivers(), CPLCheckForFile(), and VSISatExLRecursive().

13.11.2.34 int VSISatExL ( const char \* *pszFilename* )

Delete a file.

Deletes a file object from the file system.

This method goes through the VSISatExL virtualization and may work on unusual filesystems such as in memory.

Analog of the POSIX unlink() function.

## Parameters

<i>pszFilename</i>	the path of the file to be deleted. UTF-8 encoded.
--------------------	--

## Returns

0 on success or -1 on an error.

Referenced by CPLUnlinkTree().

## 13.12 ogr\_api.h File Reference

```
#include "cpl_progress.h"
#include "ogr_core.h"
```

## Functions

- OGRErr **OGR\_G\_CreateFromWkb** (unsigned char \*, OGRSpatialReferenceH, OGRGeometryH \*, int)  
*Create a geometry object of the appropriate type from its well known binary representation.*
- OGRErr **OGR\_G\_CreateFromWkt** (char \*\*, OGRSpatialReferenceH, OGRGeometryH \*)  
*Create a geometry object of the appropriate type from its well known text representation.*
- void **OGR\_G\_DestroyGeometry** (OGRGeometryH)  
*Destroy geometry object.*
- OGRGeometryH **OGR\_G\_CreateGeometry** (OGRwkbGeometryType)  
*Create an empty geometry of desired type.*
- OGRGeometryH **OGR\_G\_ApproximateArcAngles** (double dfCenterX, double dfCenterY, double dfZ, double dfPrimaryRadius, double dfSecondaryAxis, double dfRotation, double dfStartAngle, double dfEndAngle, double dfMaxAngleStepSizeDegrees)
- OGRGeometryH **OGR\_G\_ForceToPolygon** (OGRGeometryH)  
*Convert to polygon.*
- OGRGeometryH **OGR\_G\_ForceToLineString** (OGRGeometryH)



- Convert to line string.*

  - OGRGeometryH **OGR\_G\_ForceToMultiPolygon** (OGRGeometryH)
- Convert to multipolygon.*

  - OGRGeometryH **OGR\_G\_ForceToMultiPoint** (OGRGeometryH)
- Convert to multipoint.*

  - OGRGeometryH **OGR\_G\_ForceToMultiLineString** (OGRGeometryH)
- Convert to multilinestring.*

  - int **OGR\_G\_GetDimension** (OGRGeometryH)
- Get the dimension of this geometry.*

  - int **OGR\_G\_GetCoordinateDimension** (OGRGeometryH)
- Get the dimension of the coordinates in this geometry.*

  - void **OGR\_G\_SetCoordinateDimension** (OGRGeometryH, int)
- Set the coordinate dimension.*

  - OGRGeometryH **OGR\_G\_Clone** (OGRGeometryH)
- Make a copy of this object.*

  - void **OGR\_G\_GetEnvelope** (OGRGeometryH, OGREnvelope \*)
- Computes and returns the bounding envelope for this geometry in the passed psEnvelope structure.*

  - void **OGR\_G\_GetEnvelope3D** (OGRGeometryH, OGREnvelope3D \*)
- Computes and returns the bounding envelope (3D) for this geometry in the passed psEnvelope structure.*

  - OGRErr **OGR\_G\_ImportFromWkb** (OGRGeometryH, unsigned char \*, int)
- Assign geometry from well known binary data.*

  - OGRErr **OGR\_G\_ExportToWkb** (OGRGeometryH, OGRwkbByteOrder, unsigned char \*)
- Convert a geometry into well known binary format.*

  - int **OGR\_G\_WkbSize** (OGRGeometryH hGeom)
- Returns size of related binary representation.*

  - OGRErr **OGR\_G\_ImportFromWkt** (OGRGeometryH, char \*\*)
- Assign geometry from well known text data.*

  - OGRErr **OGR\_G\_ExportToWkt** (OGRGeometryH, char \*\*)
- Convert a geometry into well known text format.*

  - **OGRwkbGeometryType** **OGR\_G\_GetGeometryType** (OGRGeometryH)
- Fetch geometry type.*

  - const char \* **OGR\_G\_GetGeometryName** (OGRGeometryH)
- Fetch WKT name for geometry type.*

  - void **OGR\_G\_DumpReadable** (OGRGeometryH, FILE \*, const char \*)
- Dump geometry in well known text format to indicated output file.*

  - void **OGR\_G\_FlattenTo2D** (OGRGeometryH)
- Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.*

  - void **OGR\_G\_CloseRings** (OGRGeometryH)
- Force rings to be closed.*

  - OGRGeometryH **OGR\_G\_CreateFromGML** (const char \*)
- Create geometry from GML.*

  - char \* **OGR\_G\_ExportToGML** (OGRGeometryH)
- Convert a geometry into GML format.*

  - char \* **OGR\_G\_ExportToGMLEx** (OGRGeometryH, char \*\*papszOptions)
- Convert a geometry into GML format.*

  - char \* **OGR\_G\_ExportToKML** (OGRGeometryH, const char \*pszAltitudeMode)
- Convert a geometry into KML format.*

  - char \* **OGR\_G\_ExportToJson** (OGRGeometryH)
- Convert a geometry into GeoJSON format.*

  - char \* **OGR\_G\_ExportToJsonEx** (OGRGeometryH, char \*\*papszOptions)
- Convert a geometry into GeoJSON format.*

- void **OGR\_G\_AssignSpatialReference** (OGRGeometryH, OGRSpatialReferenceH)  
*Assign spatial reference to this object.*
- OGRSpatialReferenceH **OGR\_G\_GetSpatialReference** (OGRGeometryH)  
*Returns spatial reference system for geometry.*
- OGRErr **OGR\_G\_Transform** (OGRGeometryH, OGRCoordinateTransformationH)  
*Apply arbitrary coordinate transformation to geometry.*
- OGRErr **OGR\_G\_TransformTo** (OGRGeometryH, OGRSpatialReferenceH)  
*Transform geometry to new spatial reference system.*
- OGRGeometryH **OGR\_G\_Simplify** (OGRGeometryH hThis, double tolerance)  
*Compute a simplified geometry.*
- OGRGeometryH **OGR\_G\_SimplifyPreserveTopology** (OGRGeometryH hThis, double tolerance)  
*Simplify the geometry while preserving topology.*
- void **OGR\_G\_Segmentize** (OGRGeometryH hGeom, double dfMaxLength)  
*Modify the geometry such it has no segment longer then the given distance.*
- int **OGR\_G\_Intersects** (OGRGeometryH, OGRGeometryH)  
*Do these features intersect?*
- int **OGR\_G\_Equals** (OGRGeometryH, OGRGeometryH)  
*Returns TRUE if two geometries are equivalent.*
- int **OGR\_G\_Disjoint** (OGRGeometryH, OGRGeometryH)  
*Test for disjointness.*
- int **OGR\_G\_Touches** (OGRGeometryH, OGRGeometryH)  
*Test for touching.*
- int **OGR\_G\_Crosses** (OGRGeometryH, OGRGeometryH)  
*Test for crossing.*
- int **OGR\_G\_Within** (OGRGeometryH, OGRGeometryH)  
*Test for containment.*
- int **OGR\_G\_Contains** (OGRGeometryH, OGRGeometryH)  
*Test for containment.*
- int **OGR\_G\_Overlaps** (OGRGeometryH, OGRGeometryH)  
*Test for overlap.*
- OGRGeometryH **OGR\_G\_Boundary** (OGRGeometryH)  
*Compute boundary.*
- OGRGeometryH **OGR\_G\_ConvexHull** (OGRGeometryH)  
*Compute convex hull.*
- OGRGeometryH **OGR\_G\_Buffer** (OGRGeometryH, double, int)  
*Compute buffer of geometry.*
- OGRGeometryH **OGR\_G\_Intersection** (OGRGeometryH, OGRGeometryH)  
*Compute intersection.*
- OGRGeometryH **OGR\_G\_Union** (OGRGeometryH, OGRGeometryH)  
*Compute union.*
- OGRGeometryH **OGR\_G\_UnionCascaded** (OGRGeometryH)  
*Compute union using cascading.*
- OGRGeometryH **OGR\_G\_PointOnSurface** (OGRGeometryH)  
*Returns a point guaranteed to lie on the surface.*
- OGRGeometryH **OGR\_G\_Difference** (OGRGeometryH, OGRGeometryH)  
*Compute difference.*
- OGRGeometryH **OGR\_G\_SymDifference** (OGRGeometryH, OGRGeometryH)  
*Compute symmetric difference.*
- double **OGR\_G\_Distance** (OGRGeometryH, OGRGeometryH)  
*Compute distance between two geometries.*
- double **OGR\_G\_Length** (OGRGeometryH)

- Compute length of a geometry.*

  - double **OGR\_G\_Area** (OGRGeometryH)
- Compute geometry area.*

  - int **OGR\_G\_Centroid** (OGRGeometryH, OGRGeometryH)
- Compute the geometry centroid.*

  - void **OGR\_G\_Empty** (OGRGeometryH)
- Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.*

  - int **OGR\_G\_IsEmpty** (OGRGeometryH)
- Test if the geometry is empty.*

  - int **OGR\_G\_IsValid** (OGRGeometryH)
- Test if the geometry is valid.*

  - int **OGR\_G\_IsSimple** (OGRGeometryH)
- Returns TRUE if the geometry is simple.*

  - int **OGR\_G\_IsRing** (OGRGeometryH)
- Test if the geometry is a ring.*

  - OGRGeometryH **OGR\_G\_Polygonize** (OGRGeometryH)
- Polygonizes a set of sparse edges.*

  - OGRGeometryH **OGR\_G\_SymmetricDifference** (OGRGeometryH, OGRGeometryH) CPL\_WARN\_DEPRECATED("Non standard method. Use **OGR\_G\_SymDifference**() instead")
- Compute symmetric difference (deprecated)*

  - double **OGR\_G\_GetArea** (OGRGeometryH) CPL\_WARN\_DEPRECATED("Non standard method. Use **OGR\_G\_Area**() instead")
- Compute geometry area (deprecated)*

  - OGRGeometryH **OGR\_G\_GetBoundary** (OGRGeometryH) CPL\_WARN\_DEPRECATED("Non standard method. Use **OGR\_G\_Boundary**() instead")
- Compute boundary (deprecated)*

  - int **OGR\_G\_GetPointCount** (OGRGeometryH)
- Fetch number of points from a geometry.*

  - int **OGR\_G\_GetPoints** (OGRGeometryH hGeom, void \*pabyX, int nXStride, void \*pabyY, int nYStride, void \*pabyZ, int nZStride)
- Returns all points of line string.*

  - double **OGR\_G\_GetX** (OGRGeometryH, int)
- Fetch the x coordinate of a point from a geometry.*

  - double **OGR\_G\_GetY** (OGRGeometryH, int)
- Fetch the y coordinate of a point from a geometry.*

  - double **OGR\_G\_GetZ** (OGRGeometryH, int)
- Fetch the z coordinate of a point from a geometry.*

  - void **OGR\_G\_GetPoint** (OGRGeometryH, int iPoint, double \*, double \*, double \*)
- Fetch a point in line string or a point geometry.*

  - void **OGR\_G\_SetPoint** (OGRGeometryH, int iPoint, double, double, double)
- Set the location of a vertex in a point or linestring geometry.*

  - void **OGR\_G\_SetPoint\_2D** (OGRGeometryH, int iPoint, double, double)
- Set the location of a vertex in a point or linestring geometry.*

  - void **OGR\_G\_AddPoint** (OGRGeometryH, double, double, double)
- Add a point to a geometry (line string or point).*

  - void **OGR\_G\_AddPoint\_2D** (OGRGeometryH, double, double)
- Add a point to a geometry (line string or point).*

  - int **OGR\_G\_GetGeometryCount** (OGRGeometryH)
- Fetch the number of elements in a geometry or number of geometries in container.*

  - OGRGeometryH **OGR\_G\_GetGeometryRef** (OGRGeometryH, int)

- Fetch geometry from a geometry container.*
- OGRErr **OGR\_G\_AddGeometry** (OGRGeometryH, OGRGeometryH)
  - Add a geometry to a geometry container.*
- OGRErr **OGR\_G\_AddGeometryDirectly** (OGRGeometryH, OGRGeometryH)
  - Add a geometry directly to an existing geometry container.*
- OGRErr **OGR\_G\_RemoveGeometry** (OGRGeometryH, int, int)
  - Remove a geometry from an exiting geometry container.*
- OGRGeometryH **OGRBuildPolygonFromEdges** (OGRGeometryH hLinesAsCollection, int bBestEffort, int bAutoClose, double dfTolerance, OGRErr \*peErr)
- OGRErr **OGRSetGenerate\_DB2\_V72\_BYTE\_ORDER** (int bGenerate\_DB2\_V72\_BYTE\_ORDER)
  - Special entry point to enable the hack for generating DB2 V7.2 style WKB.*
- OGRFieldDefnH **OGR\_Fld\_Create** (const char \*, **OGRFieldType**) CPL\_WARN\_UNUSED\_RESULT
  - Create a new field definition.*
- void **OGR\_Fld\_Destroy** (OGRFieldDefnH)
  - Destroy a field definition.*
- void **OGR\_Fld\_SetName** (OGRFieldDefnH, const char \*)
  - Reset the name of this field.*
- const char \* **OGR\_Fld\_GetNameRef** (OGRFieldDefnH)
  - Fetch name of this field.*
- **OGRFieldType** **OGR\_Fld\_GetType** (OGRFieldDefnH)
  - Fetch type of this field.*
- void **OGR\_Fld\_SetType** (OGRFieldDefnH, **OGRFieldType**)
  - Set the type of this field. This should never be done to an **OGRFieldDefn** (p. ??) that is already part of an **OGR-FeatureDefn** (p. ??).*
- **OGRJustification** **OGR\_Fld\_GetJustify** (OGRFieldDefnH)
  - Get the justification for this field.*
- void **OGR\_Fld\_SetJustify** (OGRFieldDefnH, **OGRJustification**)
  - Set the justification for this field.*
- int **OGR\_Fld\_GetWidth** (OGRFieldDefnH)
  - Get the formatting width for this field.*
- void **OGR\_Fld\_SetWidth** (OGRFieldDefnH, int)
  - Set the formatting width for this field in characters.*
- int **OGR\_Fld\_GetPrecision** (OGRFieldDefnH)
  - Get the formatting precision for this field. This should normally be zero for fields of types other than OFTReal.*
- void **OGR\_Fld\_SetPrecision** (OGRFieldDefnH, int)
  - Set the formatting precision for this field in characters.*
- void **OGR\_Fld\_Set** (OGRFieldDefnH, const char \*, **OGRFieldType**, int, int, **OGRJustification**)
  - Set defining parameters for a field in one call.*
- int **OGR\_Fld\_IsIgnored** (OGRFieldDefnH hDefn)
  - Return whether this field should be omitted when fetching features.*
- void **OGR\_Fld\_SetIgnored** (OGRFieldDefnH hDefn, int)
  - Set whether this field should be omitted when fetching features.*
- const char \* **OGR\_GetFieldTypeName** (**OGRFieldType**)
  - Fetch human readable name for a field type.*
- OGRFeatureDefnH **OGR\_FD\_Create** (const char \*) CPL\_WARN\_UNUSED\_RESULT
  - Create a new feature definition object to hold the field definitions.*
- void **OGR\_FD\_Destroy** (OGRFeatureDefnH)
  - Destroy a feature definition object and release all memory associated with it.*
- void **OGR\_FD\_Release** (OGRFeatureDefnH)
  - Drop a reference, and destroy if unreferenced.*
- const char \* **OGR\_FD\_GetName** (OGRFeatureDefnH)

- Get name of the **OGRFeatureDefn** (p. ??) passed as an argument.*
- int **OGR\_FD\_GetFieldCount** (OGRFeatureDefnH)
  - Fetch number of fields on the passed feature definition.*
- OGRFieldDefnH **OGR\_FD\_GetFieldDefn** (OGRFeatureDefnH, int)
  - Fetch field definition of the passed feature definition.*
- int **OGR\_FD\_GetFieldIndex** (OGRFeatureDefnH, const char \*)
  - Find field by name.*
- void **OGR\_FD\_AddFieldDefn** (OGRFeatureDefnH, OGRFieldDefnH)
  - Add a new field definition to the passed feature definition.*
- OGRErr **OGR\_FD\_DeleteFieldDefn** (OGRFeatureDefnH hDefn, int iField)
  - Delete an existing field definition.*
- **OGRwkbGeometryType** **OGR\_FD\_GetGeomType** (OGRFeatureDefnH)
  - Fetch the geometry base type of the passed feature definition.*
- void **OGR\_FD\_SetGeomType** (OGRFeatureDefnH, **OGRwkbGeometryType**)
  - Assign the base geometry type for the passed layer (the same as the feature definition).*
- int **OGR\_FD\_IsGeometryIgnored** (OGRFeatureDefnH)
  - Determine whether the geometry can be omitted when fetching features.*
- void **OGR\_FD\_SetGeometryIgnored** (OGRFeatureDefnH, int)
  - Set whether the geometry can be omitted when fetching features.*
- int **OGR\_FD\_IsStyleIgnored** (OGRFeatureDefnH)
  - Determine whether the style can be omitted when fetching features.*
- void **OGR\_FD\_SetStyleIgnored** (OGRFeatureDefnH, int)
  - Set whether the style can be omitted when fetching features.*
- int **OGR\_FD\_Reference** (OGRFeatureDefnH)
  - Increments the reference count by one.*
- int **OGR\_FD\_Dereference** (OGRFeatureDefnH)
  - Decrements the reference count by one.*
- int **OGR\_FD\_GetReferenceCount** (OGRFeatureDefnH)
  - Fetch current reference count.*
- OGRFeatureH **OGR\_F\_Create** (OGRFeatureDefnH) CPL\_WARN\_UNUSED\_RESULT
  - Feature factory.*
- void **OGR\_F\_Destroy** (OGRFeatureH)
  - Destroy feature.*
- OGRFeatureDefnH **OGR\_F\_GetDefnRef** (OGRFeatureH)
  - Fetch feature definition.*
- OGRErr **OGR\_F\_SetGeometryDirectly** (OGRFeatureH, OGRGeometryH)
  - Set feature geometry.*
- OGRErr **OGR\_F\_SetGeometry** (OGRFeatureH, OGRGeometryH)
  - Set feature geometry.*
- OGRGeometryH **OGR\_F\_GetGeometryRef** (OGRFeatureH)
  - Fetch an handle to feature geometry.*
- OGRGeometryH **OGR\_F\_StealGeometry** (OGRFeatureH)
  - Take away ownership of geometry.*
- OGRFeatureH **OGR\_F\_Clone** (OGRFeatureH)
  - Duplicate feature.*
- int **OGR\_F\_Equal** (OGRFeatureH, OGRFeatureH)
  - Test if two features are the same.*
- int **OGR\_F\_GetFieldCount** (OGRFeatureH)
  - Fetch number of fields on this feature This will always be the same as the field count for the **OGRFeatureDefn** (p. ??).*
- OGRFieldDefnH **OGR\_F\_GetFieldDefnRef** (OGRFeatureH, int)
  - Fetch definition for this field.*

- int **OGR\_F\_GetFieldIndex** (OGRFeatureH, const char \*)  
*Fetch the field index given field name.*
- int **OGR\_F\_IsFieldSet** (OGRFeatureH, int)  
*Test if a field has ever been assigned a value or not.*
- void **OGR\_F\_UnsetField** (OGRFeatureH, int)  
*Clear a field, marking it as unset.*
- **OGRField** \* **OGR\_F\_GetRawFieldRef** (OGRFeatureH, int)  
*Fetch an handle to the internal field value given the index.*
- int **OGR\_F\_GetFieldAsInteger** (OGRFeatureH, int)  
*Fetch field value as integer.*
- double **OGR\_F\_GetFieldAsDouble** (OGRFeatureH, int)  
*Fetch field value as a double.*
- const char \* **OGR\_F\_GetFieldAsString** (OGRFeatureH, int)  
*Fetch field value as a string.*
- const int \* **OGR\_F\_GetFieldAsIntegerList** (OGRFeatureH, int, int \*)  
*Fetch field value as a list of integers.*
- const double \* **OGR\_F\_GetFieldAsDoubleList** (OGRFeatureH, int, int \*)  
*Fetch field value as a list of doubles.*
- char \*\* **OGR\_F\_GetFieldAsStringList** (OGRFeatureH, int)  
*Fetch field value as a list of strings.*
- GByte \* **OGR\_F\_GetFieldAsBinary** (OGRFeatureH, int, int \*)  
*Fetch field value as binary.*
- int **OGR\_F\_GetFieldAsDateTime** (OGRFeatureH, int, int \*, int \*, int \*, int \*, int \*, int \*, int \*)  
*Fetch field value as date and time.*
- void **OGR\_F\_SetFieldInteger** (OGRFeatureH, int, int)  
*Set field to integer value.*
- void **OGR\_F\_SetFieldDouble** (OGRFeatureH, int, double)  
*Set field to double value.*
- void **OGR\_F\_SetFieldString** (OGRFeatureH, int, const char \*)  
*Set field to string value.*
- void **OGR\_F\_SetFieldIntegerList** (OGRFeatureH, int, int, int \*)  
*Set field to list of integers value.*
- void **OGR\_F\_SetFieldDoubleList** (OGRFeatureH, int, int, double \*)  
*Set field to list of doubles value.*
- void **OGR\_F\_SetFieldStringList** (OGRFeatureH, int, char \*\*)  
*Set field to list of strings value.*
- void **OGR\_F\_SetFieldRaw** (OGRFeatureH, int, **OGRField** \*)  
*Set field.*
- void **OGR\_F\_SetFieldBinary** (OGRFeatureH, int, int, GByte \*)  
*Set field to binary data.*
- void **OGR\_F\_SetFieldDateTime** (OGRFeatureH, int, int, int, int, int, int, int, int)  
*Set field to datetime.*
- long **OGR\_F\_GetFID** (OGRFeatureH)  
*Get feature identifier.*
- OGRErr **OGR\_F\_SetFID** (OGRFeatureH, long)  
*Set the feature identifier.*
- void **OGR\_F\_DumpReadable** (OGRFeatureH, FILE \*)  
*Dump this feature in a human readable form.*
- OGRErr **OGR\_F\_SetFrom** (OGRFeatureH, OGRFeatureH, int)  
*Set one feature from another.*
- OGRErr **OGR\_F\_SetFromWithMap** (OGRFeatureH, OGRFeatureH, int, int \*)

- Set one feature from another.*

  - const char \* **OGR\_F\_GetStyleString** (OGRFeatureH)

*Fetch style string for this feature.*
- void **OGR\_F\_SetStyleString** (OGRFeatureH, const char \*)

*Set feature style string. This method operate exactly as **OGR\_F\_SetStyleStringDirectly()** (p. ??) except that it does not assume ownership of the passed string, but instead makes a copy of it.*
- void **OGR\_F\_SetStyleStringDirectly** (OGRFeatureH, char \*)

*Set feature style string. This method operate exactly as **OGR\_F\_SetStyleString()** (p. ??) except that it assumes ownership of the passed string.*
- const char \* **OGR\_L\_GetName** (OGRLayerH)

*Return the layer name.*
- **OGRwkbGeometryType** **OGR\_L\_GetGeomType** (OGRLayerH)

*Return the layer geometry type.*
- OGRGeometryH **OGR\_L\_GetSpatialFilter** (OGRLayerH)

*This function returns the current spatial filter for this layer.*
- void **OGR\_L\_SetSpatialFilter** (OGRLayerH, OGRGeometryH)

*Set a new spatial filter.*
- void **OGR\_L\_SetSpatialFilterRect** (OGRLayerH, double, double, double, double)

*Set a new rectangular spatial filter.*
- OGRErr **OGR\_L\_SetAttributeFilter** (OGRLayerH, const char \*)

*Set a new attribute query.*
- void **OGR\_L\_ResetReading** (OGRLayerH)

*Reset feature reading to start on the first feature.*
- OGRFeatureH **OGR\_L\_GetNextFeature** (OGRLayerH)

*Fetch the next available feature from this layer.*
- OGRErr **OGR\_L\_SetNextByIndex** (OGRLayerH, long)

*Move read cursor to the nIndex'th feature in the current resultset.*
- OGRFeatureH **OGR\_L\_GetFeature** (OGRLayerH, long)

*Fetch a feature by its identifier.*
- OGRErr **OGR\_L\_SetFeature** (OGRLayerH, OGRFeatureH)

*Rewrite an existing feature.*
- OGRErr **OGR\_L\_CreateFeature** (OGRLayerH, OGRFeatureH)

*Create and write a new feature within a layer.*
- OGRErr **OGR\_L\_DeleteFeature** (OGRLayerH, long)

*Delete feature from layer.*
- OGRFeatureDefnH **OGR\_L\_GetLayerDefn** (OGRLayerH)

*Fetch the schema information for this layer.*
- OGRSpatialReferenceH **OGR\_L\_GetSpatialRef** (OGRLayerH)

*Fetch the spatial reference system for this layer.*
- int **OGR\_L\_GetFeatureCount** (OGRLayerH, int)

*Fetch the feature count in this layer.*
- OGRErr **OGR\_L\_GetExtent** (OGRLayerH, **OGREnvelope** \*, int)

*Fetch the extent of this layer.*
- int **OGR\_L\_TestCapability** (OGRLayerH, const char \*)

*Test if this layer supported the named capability.*
- OGRErr **OGR\_L\_CreateField** (OGRLayerH, OGRFieldDefnH, int)

*Create a new field on a layer.*
- OGRErr **OGR\_L\_DeleteField** (OGRLayerH, int iField)

*Create a new field on a layer.*
- OGRErr **OGR\_L\_ReorderFields** (OGRLayerH, int \*panMap)

*Reorder all the fields of a layer.*

- OGRErr **OGR\_L\_ReorderField** (OGRLayerH, int iOldFieldPos, int iNewFieldPos)  
*Reorder an existing field on a layer.*
- OGRErr **OGR\_L\_AlterFieldDefn** (OGRLayerH, int iField, OGRFieldDefnH hNewFieldDefn, int nFlags)  
*Alter the definition of an existing field on a layer.*
- OGRErr **OGR\_L\_StartTransaction** (OGRLayerH)  
*For datasources which support transactions, StartTransaction creates a transaction.*
- OGRErr **OGR\_L\_CommitTransaction** (OGRLayerH)  
*For datasources which support transactions, CommitTransaction commits a transaction.*
- OGRErr **OGR\_L\_RollbackTransaction** (OGRLayerH)  
*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.*
- OGRErr **OGR\_L\_SyncToDisk** (OGRLayerH)  
*Flush pending changes to disk.*
- const char \* **OGR\_L\_GetFIDColumn** (OGRLayerH)  
*This method returns the name of the underlying database column being used as the FID column, or "" if not supported.*
- const char \* **OGR\_L\_GetGeometryColumn** (OGRLayerH)  
*This method returns the name of the underlying database column being used as the geometry column, or "" if not supported.*
- OGRErr **OGR\_L\_SetIgnoredFields** (OGRLayerH, const char \*\*)  
*Set which fields can be omitted when retrieving features from the layer.*
- OGRErr **OGR\_L\_Intersection** (OGRLayerH, OGRLayerH, OGRLayerH, char \*\*, GDALProgressFunc, void \*)  
*Intersection of two layers.*
- OGRErr **OGR\_L\_Union** (OGRLayerH, OGRLayerH, OGRLayerH, char \*\*, GDALProgressFunc, void \*)  
*Union of two layers.*
- OGRErr **OGR\_L\_SymDifference** (OGRLayerH, OGRLayerH, OGRLayerH, char \*\*, GDALProgressFunc, void \*)  
*Symmetrical difference of two layers.*
- OGRErr **OGR\_L\_Identity** (OGRLayerH, OGRLayerH, OGRLayerH, char \*\*, GDALProgressFunc, void \*)  
*Identify the features of this layer with the ones from the identity layer.*
- OGRErr **OGR\_L\_Update** (OGRLayerH, OGRLayerH, OGRLayerH, char \*\*, GDALProgressFunc, void \*)  
*Update this layer with features from the update layer.*
- OGRErr **OGR\_L\_Clip** (OGRLayerH, OGRLayerH, OGRLayerH, char \*\*, GDALProgressFunc, void \*)  
*Clip off areas that are not covered by the method layer.*
- OGRErr **OGR\_L\_Erase** (OGRLayerH, OGRLayerH, OGRLayerH, char \*\*, GDALProgressFunc, void \*)  
*Remove areas that are covered by the method layer.*
- void **OGR\_DS\_Destroy** (OGRDataSourceH)  
*Closes opened datasource and releases allocated resources.*
- const char \* **OGR\_DS\_GetName** (OGRDataSourceH)  
*Returns the name of the data source.*
- int **OGR\_DS\_GetLayerCount** (OGRDataSourceH)  
*Get the number of layers in this data source.*
- OGRLayerH **OGR\_DS\_GetLayer** (OGRDataSourceH, int)  
*Fetch a layer by index.*
- OGRLayerH **OGR\_DS\_GetLayerByName** (OGRDataSourceH, const char \*)  
*Fetch a layer by name.*
- OGRErr **OGR\_DS\_DeleteLayer** (OGRDataSourceH, int)  
*Delete the indicated layer from the datasource.*
- OGRSFDriverH **OGR\_DS\_GetDriver** (OGRDataSourceH)  
*Returns the driver that the dataset was opened with.*



- OGRLayerH **OGR\_DS\_CreateLayer** (OGRDataSourceH, const char \*, OGRSpatialReferenceH, **OGRwkbGeometryType**, char \*\*)  
*This function attempts to create a new layer on the data source with the indicated name, coordinate system, geometry type.*
- OGRLayerH **OGR\_DS\_CopyLayer** (OGRDataSourceH, OGRLayerH, const char \*, char \*\*)  
*Duplicate an existing layer.*
- int **OGR\_DS\_TestCapability** (OGRDataSourceH, const char \*)  
*Test if capability is available.*
- OGRLayerH **OGR\_DS\_ExecuteSQL** (OGRDataSourceH, const char \*, OGRGeometryH, const char \*)  
*Execute an SQL statement against the data store.*
- void **OGR\_DS\_ReleaseResultSet** (OGRDataSourceH, OGRLayerH)  
*Release results of **OGR\_DS\_ExecuteSQL()** (p. ??).*
- OGRErr **OGR\_DS\_SyncToDisk** (OGRDataSourceH)  
*Flush pending changes to disk.*
- const char \* **OGR\_Dr\_GetName** (OGRSFDriverH)  
*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".*
- OGRDataSourceH **OGR\_Dr\_Open** (OGRSFDriverH, const char \*, int) CPL\_WARN\_UNUSED\_RESULT  
*Attempt to open file with this driver.*
- int **OGR\_Dr\_TestCapability** (OGRSFDriverH, const char \*)  
*Test if capability is available.*
- OGRDataSourceH **OGR\_Dr\_CreateDataSource** (OGRSFDriverH, const char \*, char \*\*) CPL\_WARN\_UNUSED\_RESULT  
*This function attempts to create a new data source based on the passed driver.*
- OGRDataSourceH **OGR\_Dr\_CopyDataSource** (OGRSFDriverH, OGRDataSourceH, const char \*, char \*\*) CPL\_WARN\_UNUSED\_RESULT  
*This function creates a new datasource by copying all the layers from the source datasource.*
- OGRErr **OGR\_Dr\_DeleteDataSource** (OGRSFDriverH, const char \*)  
*Delete a datasource.*
- OGRDataSourceH **OGROpen** (const char \*, int, OGRSFDriverH \*) CPL\_WARN\_UNUSED\_RESULT  
*Open a file / data source with one of the registered drivers.*
- OGRErr **OGRReleaseDataSource** (OGRDataSourceH)  
*Drop a reference to this datasource, and if the reference count drops to zero close (destroy) the datasource.*
- void **OGRRegisterDriver** (OGRSFDriverH)  
*Add a driver to the list of registered drivers.*
- void **OGRDeregisterDriver** (OGRSFDriverH)  
*Remove the passed driver from the list of registered drivers.*
- int **OGRGetDriverCount** (void)  
*Fetch the number of registered drivers.*
- OGRSFDriverH **OGRGetDriver** (int)  
*Fetch the indicated driver.*
- OGRSFDriverH **OGRGetDriverByName** (const char \*)  
*Fetch the indicated driver.*
- int **OGRGetOpenDSCount** (void)  
*Return the number of opened datasources.*
- OGRDataSourceH **OGRGetOpenDS** (int iDS)  
*Return the iDS th datasource opened.*
- void **OGRRegisterAll** (void)  
*Register all drivers.*
- void **OGRCleanupAll** (void)  
*Cleanup all OGR related resources.*

- OGRStyleMgrH **OGR\_SM\_Create** (OGRStyleTableH hStyleTable) CPL\_WARN\_UNUSED\_RESULT  
*OGRStyleMgr* (p. ??) factory.
- void **OGR\_SM\_Destroy** (OGRStyleMgrH hSM)  
*Destroy Style Manager.*
- const char \* **OGR\_SM\_InitFromFeature** (OGRStyleMgrH hSM, OGRFeatureH hFeat)  
*Initialize style manager from the style string of a feature.*
- int **OGR\_SM\_InitStyleString** (OGRStyleMgrH hSM, const char \*pszStyleString)  
*Initialize style manager from the style string.*
- int **OGR\_SM\_GetPartCount** (OGRStyleMgrH hSM, const char \*pszStyleString)  
*Get the number of parts in a style.*
- OGRStyleToolH **OGR\_SM\_GetPart** (OGRStyleMgrH hSM, int nPartId, const char \*pszStyleString)  
*Fetch a part (style tool) from the current style.*
- int **OGR\_SM\_AddPart** (OGRStyleMgrH hSM, OGRStyleToolH hST)  
*Add a part (style tool) to the current style.*
- int **OGR\_SM\_AddStyle** (OGRStyleMgrH hSM, const char \*pszStyleName, const char \*pszStyleString)  
*Add a style to the current style table.*
- OGRStyleToolH **OGR\_ST\_Create** (OGRSTClassId eClassId) CPL\_WARN\_UNUSED\_RESULT  
*OGRStyleTool* (p. ??) factory.
- void **OGR\_ST\_Destroy** (OGRStyleToolH hST)  
*Destroy Style Tool.*
- OGRSTClassId **OGR\_ST\_GetType** (OGRStyleToolH hST)  
*Determine type of Style Tool.*
- OGRSTUnitId **OGR\_ST\_GetUnit** (OGRStyleToolH hST)  
*Get Style Tool units.*
- void **OGR\_ST\_SetUnit** (OGRStyleToolH hST, OGRSTUnitId eUnit, double dfGroundPaperScale)  
*Set Style Tool units.*
- const char \* **OGR\_ST\_GetParamStr** (OGRStyleToolH hST, int eParam, int \*bValueIsNull)  
*Get Style Tool parameter value as string.*
- int **OGR\_ST\_GetParamNum** (OGRStyleToolH hST, int eParam, int \*bValueIsNull)  
*Get Style Tool parameter value as an integer.*
- double **OGR\_ST\_GetParamDbl** (OGRStyleToolH hST, int eParam, int \*bValueIsNull)  
*Get Style Tool parameter value as a double.*
- void **OGR\_ST\_SetParamStr** (OGRStyleToolH hST, int eParam, const char \*pszValue)  
*Set Style Tool parameter value from a string.*
- void **OGR\_ST\_SetParamNum** (OGRStyleToolH hST, int eParam, int nValue)  
*Set Style Tool parameter value from an integer.*
- void **OGR\_ST\_SetParamDbl** (OGRStyleToolH hST, int eParam, double dfValue)  
*Set Style Tool parameter value from a double.*
- const char \* **OGR\_ST\_GetStyleString** (OGRStyleToolH hST)  
*Get the style string for this Style Tool.*
- int **OGR\_ST\_GetRGBFromString** (OGRStyleToolH hST, const char \*pszColor, int \*pnRed, int \*pnGreen, int \*pnBlue, int \*pnAlpha)  
*Return the r,g,b,a components of a color encoded in #RRGGBB[AA] format.*
- OGRStyleTableH **OGR\_STBL\_Create** (void) CPL\_WARN\_UNUSED\_RESULT  
*OGRStyleTable* (p. ??) factory.
- void **OGR\_STBL\_Destroy** (OGRStyleTableH hSTBL)  
*Destroy Style Table.*
- int **OGR\_STBL\_SaveStyleTable** (OGRStyleTableH hStyleTable, const char \*pszFilename)  
*Save a style table to a file.*
- int **OGR\_STBL\_LoadStyleTable** (OGRStyleTableH hStyleTable, const char \*pszFilename)  
*Load a style table from a file.*

- const char \* **OGR\_STBL\_Find** (OGRStyleTableH hStyleTable, const char \*pszName)  
*Get a style string by name.*
- void **OGR\_STBL\_ResetStyleStringReading** (OGRStyleTableH hStyleTable)  
*Reset the next style pointer to 0.*
- const char \* **OGR\_STBL\_GetNextStyle** (OGRStyleTableH hStyleTable)  
*Get the next style string from the table.*
- const char \* **OGR\_STBL\_GetLastStyleName** (OGRStyleTableH hStyleTable)

### 13.12.1 Detailed Description

C API and defines for **OGRFeature** (p. ??), **OGRGeometry** (p. ??), and **OGRDataSource** (p. ??) related classes.

See also: **ogr\_geometry.h** (p. ??), **ogr\_feature.h** (p. ??), **ogrsf\_frmts.h** (p. ??), **ogr\_featurestyle.h** (p. ??)

### 13.12.2 Function Documentation

**13.12.2.1 OGRDataSourceH OGR\_Dr\_CopyDataSource ( OGRSFDriverH hDriver, OGRDataSourceH hSrcDS, const char \* pszNewName, char \*\* papszOptions )**

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

It is important to call **OGR\_DS\_Destroy()** (p. ??) when the datasource is no longer used to ensure that all data has been properly flushed to disk.

This function is the same as the C++ method **OGRSFDriver::CopyDataSource()** (p. ??).

#### Parameters

<i>hDriver</i>	handle to the driver on which data source creation is based.
<i>hSrcDS</i>	source datasource
<i>pszNewName</i>	the name for the new data source.
<i>papszOptions</i>	a StringList of name=value options. Options are driver specific, and driver information can be found at the following url: <a href="http://www.gdal.org/ogr/ogr_formats.html">http://www.gdal.org/ogr/ogr_formats.html</a>

#### Returns

NULL is returned on failure, or a new **OGRDataSource** (p. ??) handle on success.

References **OGRDataSource::GetDriver()**, and **OGRDataSource::SetDriver()**.

**13.12.2.2 OGRDataSourceH OGR\_Dr\_CreateDataSource ( OGRSFDriverH hDriver, const char \* pszName, char \*\* papszOptions )**

This function 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 **OGR\_DS\_Destroy()** (p. ??) when the datasource is no longer used to ensure that all data has been properly flushed to disk.

This function is the same as the C++ method **OGRSFDriver::CreateDataSource()** (p. ??).

#### Parameters

<i>hDriver</i>	handle to the driver on which data source creation is based.
<i>pszName</i>	the name for the new data source. UTF-8 encoded.
<i>papszOptions</i>	a StringList of name=value options. Options are driver specific, and driver information can be found at the following url: <a href="http://www.gdal.org/ogr/ogr_formats.html">http://www.gdal.org/ogr/ogr_formats.html</a>

#### Returns

NULL is returned on failure, or a new **OGRDataSource** (p. ??) handle on success.

References CPLDebug(), OGRSFDriver::CreateDataSource(), OGRDataSource::GetDriver(), and OGRDataSource::SetDriver().

#### 13.12.2.3 OGRErr OGR\_Dr\_DeleteDataSource ( OGRSFDriverH *hDriver*, const char \* *pszDataSource* )

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++ method **OGRSFDriver::DeleteDataSource()** (p. ??).

#### Parameters

<i>hDriver</i>	handle to the driver on which data source deletion is based.
<i>pszDataSource</i>	the name of the datasource to delete.

#### Returns

OGRERR\_NONE on success, and OGRERR\_UNSUPPORTED\_OPERATION if this is not supported by this driver.

#### 13.12.2.4 const char \* OGR\_Dr\_GetName ( OGRSFDriverH *hDriver* )

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 function is the same as the C++ method **OGRSFDriver::GetName()** (p. ??).

#### Parameters

<i>hDriver</i>	handle to the the driver to get the name from.
----------------	--

#### Returns

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

#### 13.12.2.5 OGRDataSourceH OGR\_Dr\_Open ( OGRSFDriverH *hDriver*, const char \* *pszName*, int *bUpdate* )

Attempt to open file with this driver.

This function is the same as the C++ method **OGRSFDriver::Open()** (p. ??).

## Parameters

<i>hDriver</i>	handle to the driver that is used to open file.
<i>pszName</i>	the name of the file, or data source to try and open.
<i>bUpdate</i>	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 an handle to an **OGRDataSource** (p. ??). This **OGRDataSource** (p. ??) should be closed by deleting the object when it is no longer needed.

References OGRDataSource::GetDriver(), and OGRDataSource::SetDriver().

### 13.12.2.6 int OGR\_Dr\_TestCapability ( OGRSFDriverH *hDriver*, const char \* *pszCap* )

Test if capability is available.

One of the following data source capability names can be passed into this function, 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 misspelling.

This function is the same as the C++ method **OGRSFDriver::TestCapability()** (p. ??).

## Parameters

<i>hDriver</i>	handle to the driver to test the capability against.
<i>pszCap</i>	the capability to test.

## Returns

TRUE if capability available otherwise FALSE.

### 13.12.2.7 OGRLayerH OGR\_DS\_CopyLayer ( OGRDataSourceH *hDS*, OGRLayerH *hSrcLayer*, const char \* *pszNewName*, char \*\* *papszOptions* )

Duplicate an existing layer.

This function 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 function is the same as the C++ method **OGRDataSource::CopyLayer** (p. ??)

## Parameters

<i>hDS</i>	handle to the data source where to create the new layer
<i>hSrcLayer</i>	handle to the source layer.
<i>pszNewName</i>	the name of the layer to create.

<i>papszOptions</i>	a StringList of name=value options. Options are driver specific.
---------------------	--

#### Returns

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

**13.12.2.8 OGRLayerH OGR\_DS\_CreateLayer ( OGRDataSourceH hDS, const char \* pszName, OGRSpatialReferenceH hSpatialRef, OGRwkbGeometryType eType, char \*\* papszOptions )**

This function 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.

This function is the same as the C++ method **OGRDataSource::CreateLayer()** (p. ??).

#### Parameters

<i>hDS</i>	The dataset handle.
<i>pszName</i>	the name for the new layer. This should ideally not match any existing layer on the datasource.
<i>hSpatialRef</i>	handle to the coordinate system to use for the new layer, or NULL if no coordinate system is available.
<i>eType</i>	the geometry type for the layer. Use wkbUnknown if there are no constraints on the types geometry to be written.
<i>papszOptions</i>	a StringList of name=value options. Options are driver specific, and driver information can be found at the following url: <a href="http://www.gdal.org/ogr/ogr_formats.html">http://www.gdal.org/ogr/ogr_formats.html</a>

#### Returns

NULL is returned on failure, or a new **OGRLayer** (p. ??) handle on success.

#### Example:

```
#include "ogr_sfrmts.h"
#include "cpl_string.h"

...

OGRLayerH *hLayer;
char **papszOptions;

if( OGR_DS_TestCapability( hDS, ODS_CCreateLayer ) )
{
    ...
}

papszOptions = CSLSetNameValue( papszOptions, "DIM", "2" );
hLayer = OGR_DS_CreateLayer( hDS, "NewLayer", NULL, wkbUnknown,
                             papszOptions );
CSLDestroy( papszOptions );

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

References CPLError().

**13.12.2.9 OGRErr OGR\_DS\_DeleteLayer ( OGRDataSourceH hDS, int iLayer )**

Delete the indicated layer from the datasource.

If this method is supported the ODS\_CDeleteLayer capability will test TRUE on the **OGRDataSource** (p. ??).

This method is the same as the C++ method **OGRDataSource::DeleteLayer()** (p. ??).

## Parameters

<i>hDS</i>	handle to the datasource
<i>iLayer</i>	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.

13.12.2.10 void OGR\_DS\_Destroy ( OGRDataSourceH *hDataSource* )

Closes opened datasource and releases allocated resources.

This method is the same as the C++ method **OGRDataSource::DestroyDataSource()** (p. ??).

## Parameters

<i>hDataSource</i>	handle to allocated datasource object.
--------------------	--

13.12.2.11 OGRLayerH OGR\_DS\_ExecuteSQL ( OGRDataSourceH *hDS*, const char \* *pszSQLCommand*, OGRGeometryH *hSpatialFilter*, const char \* *pszDialect* )

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 **OGR-Layer** (p. ??) handle representing a results set from the query. Note that this **OGRLayer** (p. ??) is in addition to the layers in the data store and must be destroyed with **OGR\_DS\_ReleaseResultSet()** (p. ??) before the data source is closed (destroyed).

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.

This function is the same as the C++ method **OGRDataSource::ExecuteSQL()** (p. ??);

## Parameters

<i>hDS</i>	handle to the data source on which the SQL query is executed.
<i>pszSQL-Command</i>	the SQL statement to execute.
<i>hSpatialFilter</i>	handle to a geometry which represents a spatial filter. Can be NULL.
<i>pszDialect</i>	allows control of the statement dialect. If set to NULL, the OGR SQL engine will be used, except for RDBMS drivers that will use their dedicated SQL engine, unless OGRSQL is explicitly passed as the dialect. Starting with OGR 1.10, the <code>SQLITE dialect</code> can also be used.

## Returns

an handle to a **OGRLayer** (p. ??) containing the results of the query. Deallocate with **OGR\_DS\_ReleaseResultSet()** (p. ??).

13.12.2.12 OGRSFDriverH OGR\_DS\_GetDriver ( OGRDataSourceH *hDS* )

Returns the driver that the dataset was opened with.

This method is the same as the C++ method **OGRDataSource::GetDriver()** (p. ??)

## Parameters

<i>hDS</i>	handle to the datasource
------------	--------------------------

## Returns

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

13.12.2.13 OGRLayerH OGR\_DS\_GetLayer ( OGRDataSourceH *hDS*, int *iLayer* )

Fetch a layer by index.

The returned layer remains owned by the **OGRDataSource** (p. ??) and should not be deleted by the application.

This function is the same as the C++ method **OGRDataSource::GetLayer()** (p. ??).

## Parameters

<i>hDS</i>	handle to the data source from which to get the layer.
<i>iLayer</i>	a layer number between 0 and <b>OGR_DS_GetLayerCount()</b> (p. ??)-1.

## Returns

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

13.12.2.14 OGRLayerH OGR\_DS\_GetLayerByName ( OGRDataSourceH *hDS*, const char \* *pszLayerName* )

Fetch a layer by name.

The returned layer remains owned by the **OGRDataSource** (p. ??) and should not be deleted by the application.

This function is the same as the C++ method **OGRDataSource::GetLayerByName()** (p. ??).

## Parameters

<i>hDS</i>	handle to the data source from which to get the layer.
<i>pszLayerName</i>	Layer the layer name of the layer to fetch.

## Returns

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

13.12.2.15 int OGR\_DS\_GetLayerCount ( OGRDataSourceH *hDS* )

Get the number of layers in this data source.

This function is the same as the C++ method **OGRDataSource::GetLayerCount()** (p. ??).

## Parameters

<i>hDS</i>	handle to the data source from which to get the number of layers.
------------	---

## Returns

layer count.



13.12.2.16 `const char * OGR_DS_GetName ( OGRDataSourceH hDS )`

Returns the name of the data source.

This string should be sufficient to open the data source if passed to the same **OGRSFDriver** (p. ??) 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 function is the same as the C++ method **OGRDataSource::GetName()** (p. ??).

## Parameters

<i>hDS</i>	handle to the data source to get the name from.
------------	---

## Returns

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

13.12.2.17 `void OGR_DS_ReleaseResultSet ( OGRDataSourceH hDS, OGRLayerH hLayer )`

Release results of **OGR\_DS\_ExecuteSQL()** (p. ??).

This function should only be used to deallocate OGRLayers resulting from an **OGR\_DS\_ExecuteSQL()** (p. ??) call on the same **OGRDataSource** (p. ??). Failure to deallocate a results set before destroying the **OGRDataSource** (p. ??) may cause errors.

This function is the same as the C++ method **OGRDataSource::ReleaseResultSet()** (p. ??).

## Parameters

<i>hDS</i>	an handle to the data source on which was executed an SQL query.
<i>hLayer</i>	handle to the result of a previous <b>OGR_DS_ExecuteSQL()</b> (p. ??) call.

13.12.2.18 `OGRERR OGR_DS_SyncToDisk ( OGRDataSourceH hDS )`

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 **OGR\_DS\_Destroy()** (p. ??) that will ensure all data is correctly flushed.

This method is the same as the C++ method **OGRDataSource::SyncToDisk()** (p. ??)

## Parameters

<i>hDS</i>	handle to the data source
------------	---------------------------

## Returns

OGRERR\_NONE if no error occurs (even if nothing is done) or an error code.

13.12.2.19 `int OGR_DS_TestCapability ( OGRDataSourceH hDS, const char * pszCapability )`

Test if capability is available.

One of the following data source capability names can be passed into this function, 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.

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

This function is the same as the C++ method **OGRDataSource::TestCapability()** (p. ??).

#### Parameters

<i>hDS</i>	handle to the data source against which to test the capability.
<i>pszCapability</i>	the capability to test.

#### Returns

TRUE if capability available otherwise FALSE.

13.12.2.20 `OGRFeatureH OGR_F_Clone ( OGRFeatureH hFeat )`

Duplicate feature.

The newly created feature is owned by the caller, and will have it's own reference to the **OGRFeatureDefn** (p. ??).

This function is the same as the C++ method **OGRFeature::Clone()** (p. ??).

#### Parameters

<i>hFeat</i>	handle to the feature to clone.
--------------	---------------------------------

#### Returns

an handle to the new feature, exactly matching this feature.

13.12.2.21 `OGRFeatureH OGR_F_Create ( OGRFeatureDefnH hDefn )`

Feature factory.

Note that the **OGRFeature** (p. ??) will increment the reference count of it's defining **OGRFeatureDefn** (p. ??). Destruction of the **OGRFeatureDefn** (p. ??) before destruction of all OGRFeatures that depend on it is likely to result in a crash.

This function is the same as the C++ method **OGRFeature::OGRFeature()** (p. ??).

#### Parameters

<i>hDefn</i>	handle to the feature class (layer) definition to which the feature will adhere.
--------------	--

#### Returns

an handle to the new feature object with null fields and no geometry.

**13.12.2.22 void OGR\_F\_Destroy ( OGRFeatureH *hFeat* )**

Destroy feature.

The feature is deleted, but within the context of the GDAL/OGR heap. This is necessary when higher level applications use GDAL/OGR from a DLL and they want to delete a feature created within the DLL. If the delete is done in the calling application the memory will be freed onto the application heap which is inappropriate.

This function is the same as the C++ method **OGRFeature::DestroyFeature()** (p. ??).

**Parameters**

<i>hFeat</i>	handle to the feature to destroy.
--------------	-----------------------------------

**13.12.2.23 void OGR\_F\_DumpReadable ( OGRFeatureH *hFeat*, FILE \* *fpOut* )**

Dump this feature in a human readable form.

This dumps the attributes, and geometry; however, it doesn't definition information (other than field types and names), nor does it report the geometry spatial reference system.

This function is the same as the C++ method **OGRFeature::DumpReadable()** (p. ??).

**Parameters**

<i>hFeat</i>	handle to the feature to dump.
<i>fpOut</i>	the stream to write to, such as strout.

**13.12.2.24 int OGR\_F\_Equal ( OGRFeatureH *hFeat*, OGRFeatureH *hOtherFeat* )**

Test if two features are the same.

Two features are considered equal if they share them (handle equality) same **OGRFeatureDefn** (p. ??), have the same field values, and the same geometry (as tested by **OGR\_G\_Equal()**) as well as the same feature id.

This function is the same as the C++ method **OGRFeature::Equal()** (p. ??).

**Parameters**

<i>hFeat</i>	handle to one of the feature.
<i>hOtherFeat</i>	handle to the other feature to test this one against.

**Returns**

TRUE if they are equal, otherwise FALSE.

**13.12.2.25 OGRFeatureDefnH OGR\_F\_GetDefnRef ( OGRFeatureH *hFeat* )**

Fetch feature definition.

This function is the same as the C++ method **OGRFeature::GetDefnRef()** (p. ??).

**Parameters**

<i>hFeat</i>	handle to the feature to get the feature definition from.
--------------	---

**Returns**

an handle to the feature definition object on which feature depends.

### 13.12.2.26 long OGR\_F\_GetFID ( OGRFeatureH *hFeat* )

Get feature identifier.

This function is the same as the C++ method **OGRFeature::GetFID()** (p. ??).

#### Parameters

<i>hFeat</i>	handle to the feature from which to get the feature identifier.
--------------	---

#### Returns

feature id or OGRNullFID if none has been assigned.

### 13.12.2.27 GByte\* OGR\_F\_GetFieldAsBinary ( OGRFeatureH *hFeat*, int *iField*, int \* *pnBytes* )

Fetch field value as binary.

Currently this method only works for OFTBinary fields.

This function is the same as the C++ method **OGRFeature::GetFieldAsBinary()** (p. ??).

#### Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>pnBytes</i>	location to place count of bytes returned.

#### Returns

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief.

### 13.12.2.28 int OGR\_F\_GetFieldAsDateTime ( OGRFeatureH *hFeat*, int *iField*, int \* *pnYear*, int \* *pnMonth*, int \* *pnDay*, int \* *pnHour*, int \* *pnMinute*, int \* *pnSecond*, int \* *pnTZFlag* )

Fetch field value as date and time.

Currently this method only works for OFTDate, OFTTime and OFTDateTime fields.

This function is the same as the C++ method **OGRFeature::GetFieldAsDateTime()** (p. ??).

#### Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>pnYear</i>	(including century)
<i>pnMonth</i>	(1-12)
<i>pnDay</i>	(1-31)
<i>pnHour</i>	(0-23)
<i>pnMinute</i>	(0-59)
<i>pnSecond</i>	(0-59)
<i>pnTZFlag</i>	(0=unknown, 1=localtime, 100=GMT, see data model for details)

#### Returns

TRUE on success or FALSE on failure.

### 13.12.2.29 double OGR\_F\_GetFieldAsDouble ( OGRFeatureH *hFeat*, int *iField* )

Fetch field value as a double.

OFTString features will be translated using atof(). OFTInteger fields will be cast to double. Other field types, or errors will result in a return value of zero.

This function is the same as the C++ method **OGRFeature::GetFieldAsDouble()** (p. ??).

#### Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.

#### Returns

the field value.

### 13.12.2.30 const double\* OGR\_F\_GetFieldAsDoubleList ( OGRFeatureH *hFeat*, int *iField*, int \* *pnCount* )

Fetch field value as a list of doubles.

Currently this function only works for OFTRealList fields.

This function is the same as the C++ method **OGRFeature::GetFieldAsDoubleList()** (p. ??).

#### Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>pnCount</i>	an integer to put the list count (number of doubles) into.

#### Returns

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief. If \*pnCount is zero on return the returned pointer may be NULL or non-NULL.

### 13.12.2.31 int OGR\_F\_GetFieldAsInteger ( OGRFeatureH *hFeat*, int *iField* )

Fetch field value as integer.

OFTString features will be translated using atoi(). OFTReal fields will be cast to integer. Other field types, or errors will result in a return value of zero.

This function is the same as the C++ method **OGRFeature::GetFieldAsInteger()** (p. ??).

#### Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.

#### Returns

the field value.

### 13.12.2.32 const int\* OGR\_F\_GetFieldAsIntegerList ( OGRFeatureH *hFeat*, int *iField*, int \* *pnCount* )

Fetch field value as a list of integers.

Currently this function only works for OFTIntegerList fields.

This function is the same as the C++ method **OGRFeature::GetFieldAsIntegerList()** (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>pnCount</i>	an integer to put the list count (number of integers) into.

## Returns

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief. If \*pnCount is zero on return the returned pointer may be NULL or non-NULL.

### 13.12.2.33 `const char* OGR_F_GetFieldAsString ( OGRFeatureH hFeat, int iField )`

Fetch field value as a string.

OFTReal and OFTInteger fields will be translated to string using sprintf(), but not necessarily using the established formatting rules. Other field types, or errors will result in a return value of zero.

This function is the same as the C++ method **OGRFeature::GetFieldAsString()** (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.

## Returns

the field value. This string is internal, and should not be modified, or freed. Its lifetime may be very brief.

### 13.12.2.34 `char** OGR_F_GetFieldAsStringList ( OGRFeatureH hFeat, int iField )`

Fetch field value as a list of strings.

Currently this method only works for OFTStringList fields.

The returned list is terminated by a NULL pointer. The number of elements can also be calculated using **CSLCount()** (p. ??).

This function is the same as the C++ method **OGRFeature::GetFieldAsStringList()** (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.

## Returns

the field value. This list is internal, and should not be modified, or freed. Its lifetime may be very brief.

### 13.12.2.35 `int OGR_F_GetFieldCount ( OGRFeatureH hFeat )`

Fetch number of fields on this feature This will always be the same as the field count for the **OGRFeatureDefn** (p. ??).

This function is the same as the C++ method **OGRFeature::GetFieldCount()** (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature to get the fields count from.
--------------	---

## Returns

count of fields.

13.12.2.36 OGRFieldDefnH OGR\_F\_GetFieldDefnRef ( OGRFeatureH *hFeat*, int *i* )

Fetch definition for this field.

This function is the same as the C++ method **OGRFeature::GetFieldDefnRef()** (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature on which the field is found.
<i>i</i>	the field to fetch, from 0 to GetFieldCount()-1.

## Returns

an handle to the field definition (from the **OGRFeatureDefn** (p. ??)). This is an internal reference, and should not be deleted or modified.

13.12.2.37 int OGR\_F\_GetFieldIndex ( OGRFeatureH *hFeat*, const char \* *pszName* )

Fetch the field index given field name.

This is a cover for the **OGRFeatureDefn::GetFieldIndex()** (p. ??) method.

This function is the same as the C++ method **OGRFeature::GetFieldIndex()** (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature on which the field is found.
<i>pszName</i>	the name of the field to search for.

## Returns

the field index, or -1 if no matching field is found.

13.12.2.38 OGRGeometryH OGR\_F\_GetGeometryRef ( OGRFeatureH *hFeat* )

Fetch an handle to feature geometry.

This function is the same as the C++ method **OGRFeature::GetGeometryRef()** (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature to get geometry from.
--------------	---

## Returns

an handle to internal feature geometry. This object should not be modified.

13.12.2.39 OGRField\* OGR\_F\_GetRawFieldRef ( OGRFeatureH *hFeat*, int *iField* )

Fetch an handle to the internal field value given the index.

This function is the same as the C++ method **OGRFeature::GetRawFieldRef()** (p. ??).

**Parameters**

<i>hFeat</i>	handle to the feature on which field is found.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.

**Returns**

the returned handle is to an internal data structure, and should not be freed, or modified.

**13.12.2.40 const char\* OGR\_F\_GetStyleString ( OGRFeatureH hFeat )**

Fetch style string for this feature.

Set the OGR Feature Style Specification for details on the format of this string, and **ogr\_featurestyle.h** (p. ??) for services available to parse it.

This function is the same as the C++ method **OGRFeature::GetStyleString()** (p. ??).

**Parameters**

<i>hFeat</i>	handle to the feature to get the style from.
--------------	--

**Returns**

a reference to a representation in string format, or NULL if there isn't one.

**13.12.2.41 int OGR\_F\_IsFieldSet ( OGRFeatureH hFeat, int iField )**

Test if a field has ever been assigned a value or not.

This function is the same as the C++ method **OGRFeature::IsFieldSet()** (p. ??).

**Parameters**

<i>hFeat</i>	handle to the feature on which the field is.
<i>iField</i>	the field to test.

**Returns**

TRUE if the field has been set, otherwise false.

References CPLError(), OGRFeature::GetFieldCount(), and OGRFeature::IsFieldSet().

**13.12.2.42 OGRErr OGR\_F\_SetFID ( OGRFeatureH hFeat, long nFID )**

Set the feature identifier.

For specific types of features this operation may fail on illegal features ids. Generally it always succeeds. Feature ids should be greater than or equal to zero, with the exception of OGRNullFID (-1) indicating that the feature id is unknown.

This function is the same as the C++ method **OGRFeature::SetFID()** (p. ??).

**Parameters**

<i>hFeat</i>	handle to the feature to set the feature id to.
--------------	---



<i>nFID</i>	the new feature identifier value to assign.
-------------	---

**Returns**

On success OGRERR\_NONE, or on failure some other value.

**13.12.2.43** void OGR\_F\_SetFieldBinary ( OGRFeatureH *hFeat*, int *iField*, int *nBytes*, GByte \* *pabyData* )

Set field to binary data.

This function currently on has an effect of OFTBinary fields.

This function is the same as the C++ method **OGRFeature::SetField()** (p. ??).

**Parameters**

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to set, from 0 to GetFieldCount()-1.
<i>nBytes</i>	the number of bytes in pabyData array.
<i>pabyData</i>	the data to apply.

**13.12.2.44** void OGR\_F\_SetFieldDateTime ( OGRFeatureH *hFeat*, int *iField*, int *nYear*, int *nMonth*, int *nDay*, int *nHour*, int *nMinute*, int *nSecond*, int *nTZFlag* )

Set field to datetime.

This method currently only has an effect for OFTDate, OFTTime and OFTDateTime fields.

**Parameters**

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to set, from 0 to GetFieldCount()-1.
<i>nYear</i>	(including century)
<i>nMonth</i>	(1-12)
<i>nDay</i>	(1-31)
<i>nHour</i>	(0-23)
<i>nMinute</i>	(0-59)
<i>nSecond</i>	(0-59)
<i>nTZFlag</i>	(0=unknown, 1=localtime, 100=GMT, see data model for details)

**13.12.2.45** void OGR\_F\_SetFieldDouble ( OGRFeatureH *hFeat*, int *iField*, double *dfValue* )

Set field to double value.

OFTInteger and OFTReal fields will be set directly. OFTString fields will be assigned a string representation of the value, but not necessarily taking into account formatting constraints on this field. Other field types may be unaffected.

This function is the same as the C++ method **OGRFeature::SetField()** (p. ??).

**Parameters**

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.

<i>dfValue</i>	the value to assign.
----------------	----------------------

13.12.2.46 void OGR\_F\_SetFieldDoubleList ( OGRFeatureH *hFeat*, int *iField*, int *nCount*, double \* *padfValues* )

Set field to list of doubles value.

This function currently on has an effect of OFTRealList fields.

This function is the same as the C++ method **OGRFeature::SetField()** (p. ??).

#### Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to set, from 0 to GetFieldCount()-1.
<i>nCount</i>	the number of values in the list being assigned.
<i>padfValues</i>	the values to assign.

13.12.2.47 void OGR\_F\_SetFieldInteger ( OGRFeatureH *hFeat*, int *iField*, int *nValue* )

Set field to integer value.

OFTInteger and OFTReal fields will be set directly. OFTString fields will be assigned a string representation of the value, but not necessarily taking into account formatting constraints on this field. Other field types may be unaffected.

This function is the same as the C++ method **OGRFeature::SetField()** (p. ??).

#### Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>nValue</i>	the value to assign.

13.12.2.48 void OGR\_F\_SetFieldIntegerList ( OGRFeatureH *hFeat*, int *iField*, int *nCount*, int \* *panValues* )

Set field to list of integers value.

This function currently on has an effect of OFTIntegerList fields.

This function is the same as the C++ method **OGRFeature::SetField()** (p. ??).

#### Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to set, from 0 to GetFieldCount()-1.
<i>nCount</i>	the number of values in the list being assigned.
<i>panValues</i>	the values to assign.

13.12.2.49 void OGR\_F\_SetFieldRaw ( OGRFeatureH *hFeat*, int *iField*, OGRField \* *psValue* )

Set field.

The passed value **OGRField** (p. ??) must be of exactly the same type as the target field, or an application crash may occur. The passed value is copied, and will not be affected. It remains the responsibility of the caller.

This function is the same as the C++ method **OGRFeature::SetField()** (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>pszValue</i>	handle on the value to assign.

13.12.2.50 void OGR\_F\_SetFieldString ( OGRFeatureH *hFeat*, int *iField*, const char \* *pszValue* )

Set field to string value.

OFTInteger fields will be set based on an atoi() conversion of the string. OFTReal fields will be set based on an atof() conversion of the string. Other field types may be unaffected.

This function is the same as the C++ method **OGRFeature::SetField()** (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to fetch, from 0 to GetFieldCount()-1.
<i>pszValue</i>	the value to assign.

13.12.2.51 void OGR\_F\_SetFieldStringList ( OGRFeatureH *hFeat*, int *iField*, char \*\* *papszValues* )

Set field to list of strings value.

This function currently on has an effect of OFTStringList fields.

This function is the same as the C++ method **OGRFeature::SetField()** (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature that owned the field.
<i>iField</i>	the field to set, from 0 to GetFieldCount()-1.
<i>papszValues</i>	the values to assign.

13.12.2.52 OGRErr OGR\_F\_SetFrom ( OGRFeatureH *hFeat*, OGRFeatureH *hOtherFeat*, int *bForgiving* )

Set one feature from another.

Overwrite the contents of this feature from the geometry and attributes of another. The *hOtherFeature* does not need to have the same **OGRFeatureDefn** (p. ??). Field values are copied by corresponding field names. Field types do not have to exactly match. OGR\_F\_SetField\*() function conversion rules will be applied as needed.

This function is the same as the C++ method **OGRFeature::SetFrom()** (p. ??).

## Parameters

<i>hFeat</i>	handle to the feature to set to.
<i>hOtherFeat</i>	handle to the feature from which geometry, and field values will be copied.
<i>bForgiving</i>	TRUE if the operation should continue despite lacking output fields matching some of the source fields.

## Returns

OGRERR\_NONE if the operation succeeds, even if some values are not transferred, otherwise an error code.

13.12.2.53 OGRErr OGR\_F\_SetFromWithMap ( OGRFeatureH *hFeat*, OGRFeatureH *hOtherFeat*, int *bForgiving*, int \* *panMap* )

Set one feature from another.

Overwrite the contents of this feature from the geometry and attributes of another. The `hOtherFeature` does not need to have the same **OGRFeatureDefn** (p. ??). Field values are copied according to the provided indices map. Field types do not have to exactly match. `OGR_F_SetField*()` function conversion rules will be applied as needed. This is more efficient than **OGR\_F\_SetFrom()** (p. ??) in that this doesn't lookup the fields by their names. Particularly useful when the field names don't match.

This function is the same as the C++ method **OGRFeature::SetFrom()** (p. ??).

#### Parameters

<i>hFeat</i>	handle to the feature to set to.
<i>hOtherFeat</i>	handle to the feature from which geometry, and field values will be copied.
<i>panMap</i>	Array of the indices of the destination feature's fields stored at the corresponding index of the source feature's fields. A value of -1 should be used to ignore the source's field. The array should not be NULL and be as long as the number of fields in the source feature.
<i>bForgiving</i>	TRUE if the operation should continue despite lacking output fields matching some of the source fields.

#### Returns

OGRERR\_NONE if the operation succeeds, even if some values are not transferred, otherwise an error code.

#### 13.12.2.54 OGRErr OGR\_F\_SetGeometry ( OGRFeatureH *hFeat*, OGRGeometryH *hGeom* )

Set feature geometry.

This function updates the features geometry, and operate exactly as `SetGeometryDirectly()`, except that this function does not assume ownership of the passed geometry, but instead makes a copy of it.

This function is the same as the C++ **OGRFeature::SetGeometry()** (p. ??).

#### Parameters

<i>hFeat</i>	handle to the feature on which new geometry is applied to.
<i>hGeom</i>	handle to the new geometry to apply to feature.

#### Returns

OGRERR\_NONE if successful, or OGR\_UNSUPPORTED\_GEOMETRY\_TYPE if the geometry type is illegal for the **OGRFeatureDefn** (p. ??) (checking not yet implemented).

#### 13.12.2.55 OGRErr OGR\_F\_SetGeometryDirectly ( OGRFeatureH *hFeat*, OGRGeometryH *hGeom* )

Set feature geometry.

This function updates the features geometry, and operate exactly as `SetGeometry()`, except that this function assumes ownership of the passed geometry.

This function is the same as the C++ method **OGRFeature::SetGeometryDirectly** (p. ??).

#### Parameters

<i>hFeat</i>	handle to the feature on which to apply the geometry.
<i>hGeom</i>	handle to the new geometry to apply to feature.

#### Returns

OGRERR\_NONE if successful, or OGR\_UNSUPPORTED\_GEOMETRY\_TYPE if the geometry type is illegal for the **OGRFeatureDefn** (p. ??) (checking not yet implemented).

13.12.2.56 void OGR\_F\_SetStyleString ( OGRFeatureH *hFeat*, const char \* *pszStyle* )

Set feature style string. This method operate exactly as **OGR\_F\_SetStyleStringDirectly()** (p. ??) except that it does not assume ownership of the passed string, but instead makes a copy of it.

This function is the same as the C++ method **OGRFeature::SetStyleString()** (p. ??).

#### Parameters

<i>hFeat</i>	handle to the feature to set style to.
<i>pszStyle</i>	the style string to apply to this feature, cannot be NULL.

13.12.2.57 void OGR\_F\_SetStyleStringDirectly ( OGRFeatureH *hFeat*, char \* *pszStyle* )

Set feature style string. This method operate exactly as **OGR\_F\_SetStyleString()** (p. ??) except that it assumes ownership of the passed string.

This function is the same as the C++ method **OGRFeature::SetStyleStringDirectly()** (p. ??).

#### Parameters

<i>hFeat</i>	handle to the feature to set style to.
<i>pszStyle</i>	the style string to apply to this feature, cannot be NULL.

13.12.2.58 OGRGeometryH OGR\_F\_StealGeometry ( OGRFeatureH *hFeat* )

Take away ownership of geometry.

Fetch the geometry from this feature, and clear the reference to the geometry on the feature. This is a mechanism for the application to take over ownship of the geometry from the feature without copying. Sort of an inverse to **OGR\_FSetGeometryDirectly()**.

After this call the **OGRFeature** (p. ??) will have a NULL geometry.

#### Returns

the pointer to the geometry.

13.12.2.59 void OGR\_F\_UnsetField ( OGRFeatureH *hFeat*, int *iField* )

Clear a field, marking it as unset.

This function is the same as the C++ method **OGRFeature::UnsetField()** (p. ??).

#### Parameters

<i>hFeat</i>	handle to the feature on which the field is.
<i>iField</i>	the field to unset.

13.12.2.60 void OGR\_FD\_AddFieldDefn ( OGRFeatureDefnH *hDefn*, OGRFieldDefnH *hNewField* )

Add a new field definition to the passed feature definition.

To add a new field definition to a layer definition, do not use this function directly, but use **OGR\_L\_CreateField()** (p. ??) instead.

This function should only be called while there are no **OGRFeature** (p. ??) objects in existence based on this **OGRFeatureDefn** (p. ??). The **OGRFieldDefn** (p. ??) passed in is copied, and remains the responsibility of the caller.

This function is the same as the C++ method **OGRFeatureDefn::AddFieldDefn()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the feature definition to add the field definition to.
<i>hNewField</i>	handle to the new field definition.

13.12.2.61 OGRFeatureDefnH OGR\_FD\_Create ( const char \* *pszName* )

Create a new feature definition object to hold the field definitions.

The **OGRFeatureDefn** (p. ??) maintains a reference count, but this starts at zero, and should normally be incremented by the owner.

This function is the same as the C++ method **OGRFeatureDefn::OGRFeatureDefn()** (p. ??).

## Parameters

<i>pszName</i>	the name to be assigned to this layer/class. It does not need to be unique.
----------------	---

## Returns

handle to the newly created feature definition.

13.12.2.62 OGRErr OGR\_FD\_DeleteFieldDefn ( OGRFeatureDefnH *hDefn*, int *iField* )

Delete an existing field definition.

To delete an existing field definition from a layer definition, do not use this function directly, but use **OGR\_L\_DeleteField()** (p. ??) instead.

This method should only be called while there are no **OGRFeature** (p. ??) objects in existence based on this **OGRFeatureDefn** (p. ??).

This method is the same as the C++ method **OGRFeatureDefn::DeleteFieldDefn()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the feature definition.
<i>iField</i>	the index of the field definition.

## Returns

OGRERR\_NONE in case of success.

## Since

OGR 1.9.0

13.12.2.63 int OGR\_FD\_Dereference ( OGRFeatureDefnH *hDefn* )

Decrements the reference count by one.

This function is the same as the C++ method **OGRFeatureDefn::Dereference()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the feature definition on witch <b>OGRFeature</b> (p. ??) are based on.
--------------	---

**Returns**

the updated reference count.

13.12.2.64 `void OGR_FD_Destroy ( OGRFeatureDefnH hDefn )`

Destroy a feature definition object and release all memory associated with it.

This function is the same as the C++ method `OGRFeatureDefn::~OGRFeatureDefn()`.

**Parameters**

<i>hDefn</i>	handle to the feature definition to be destroyed.
--------------	---

13.12.2.65 `int OGR_FD_GetFieldCount ( OGRFeatureDefnH hDefn )`

Fetch number of fields on the passed feature definition.

This function is the same as the C++ **OGRFeatureDefn::GetFieldCount()** (p. ??).

**Parameters**

<i>hDefn</i>	handle to the feature definition to get the fields count from.
--------------	--

**Returns**

count of fields.

13.12.2.66 `OGRFieldDefnH OGR_FD_GetFieldDefn ( OGRFeatureDefnH hDefn, int iField )`

Fetch field definition of the passed feature definition.

This function is the same as the C++ method **OGRFeatureDefn::GetFieldDefn()** (p. ??).

Starting with GDAL 1.7.0, this method will also issue an error if the index is not valid.

**Parameters**

<i>hDefn</i>	handle to the feature definition to get the field definition from.
<i>iField</i>	the field to fetch, between 0 and GetFieldCount()-1.

**Returns**

an handle to an internal field definition object or NULL if invalid index. This object should not be modified or freed by the application.

13.12.2.67 `int OGR_FD_GetFieldIndex ( OGRFeatureDefnH hDefn, const char * pszFieldName )`

Find field by name.

The field index of the first field matching the passed field name (case insensitively) is returned.

This function is the same as the C++ method **OGRFeatureDefn::GetFieldIndex** (p. ??).



## Parameters

<i>hDefn</i>	handle to the feature definition to get field index from.
<i>pszFieldName</i>	the field name to search for.

## Returns

the field index, or -1 if no match found.

#### 13.12.2.68 **OGRwkbGeometryType** OGR\_FD\_GetGeomType ( OGRFeatureDefnH *hDefn* )

Fetch the geometry base type of the passed feature definition.

This function is the same as the C++ method **OGRFeatureDefn::GetGeomType()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the feature definition to get the geometry type from.
--------------	---

## Returns

the base type for all geometry related to this definition.

#### 13.12.2.69 **const char\*** OGR\_FD\_GetName ( OGRFeatureDefnH *hDefn* )

Get name of the **OGRFeatureDefn** (p. ??) passed as an argument.

This function is the same as the C++ method **OGRFeatureDefn::GetName()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the feature definition to get the name from.
--------------	--

## Returns

the name. This name is internal and should not be modified, or freed.

#### 13.12.2.70 **int** OGR\_FD\_GetReferenceCount ( OGRFeatureDefnH *hDefn* )

Fetch current reference count.

This function is the same as the C++ method **OGRFeatureDefn::GetReferenceCount()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the feature definition on which <b>OGRFeature</b> (p. ??) are based on.
--------------	---

## Returns

the current reference count.

#### 13.12.2.71 **int** OGR\_FD\_IsGeometryIgnored ( OGRFeatureDefnH *hDefn* )

Determine whether the geometry can be omitted when fetching features.

This function is the same as the C++ method **OGRFeatureDefn::IsGeometryIgnored()** (p. ??).

**Parameters**

<i>hDefn</i>	handle to the feature definition on which <b>OGRFeature</b> (p. ??) are based on.
--------------	---

**Returns**

ignore state

**13.12.2.72 int OGR\_FD\_IsStyleIgnored ( OGRFeatureDefnH hDefn )**

Determine whether the style can be omitted when fetching features.

This function is the same as the C++ method **OGRFeatureDefn::IsStyleIgnored()** (p. ??).

**Parameters**

<i>hDefn</i>	handle to the feature definition on which <b>OGRFeature</b> (p. ??) are based on.
--------------	---

**Returns**

ignore state

**13.12.2.73 int OGR\_FD\_Reference ( OGRFeatureDefnH hDefn )**

Increments the reference count by one.

The reference count is used keep track of the number of **OGRFeature** (p. ??) objects referencing this definition.

This function is the same as the C++ method **OGRFeatureDefn::Reference()** (p. ??).

**Parameters**

<i>hDefn</i>	handle to the feature definition on which <b>OGRFeature</b> (p. ??) are based on.
--------------	---

**Returns**

the updated reference count.

**13.12.2.74 void OGR\_FD\_Release ( OGRFeatureDefnH hDefn )**

Drop a reference, and destroy if unreferenced.

This function is the same as the C++ method **OGRFeatureDefn::Release()** (p. ??).

**Parameters**

<i>hDefn</i>	handle to the feature definition to be released.
--------------	--

**13.12.2.75 void OGR\_FD\_SetGeometryIgnored ( OGRFeatureDefnH hDefn, int bIgnore )**

Set whether the geometry can be omitted when fetching features.

This function is the same as the C++ method **OGRFeatureDefn::SetGeometryIgnored()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the feature definition on witch <b>OGRFeature</b> (p. ??) are based on.
<i>blgnore</i>	ignore state

13.12.2.76 void OGR\_FD\_SetGeomType ( OGRFeatureDefnH *hDefn*, OGRwkbGeometryType *eType* )

Assign the base geometry type for the passed layer (the same as the feature definition).

All geometry objects using this type must be of the defined type or a derived type. The default upon creation is wkbUnknown which allows for any geometry type. The geometry type should generally not be changed after any OGRFeatures have been created against this definition.

This function is the same as the C++ method **OGRFeatureDefn::SetGeomType()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the layer or feature definition to set the geometry type to.
<i>eType</i>	the new type to assign.

13.12.2.77 void OGR\_FD\_SetStyleIgnored ( OGRFeatureDefnH *hDefn*, int *blgnore* )

Set whether the style can be omitted when fetching features.

This function is the same as the C++ method **OGRFeatureDefn::SetStyleIgnored()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the feature definition on witch <b>OGRFeature</b> (p. ??) are based on.
<i>blgnore</i>	ignore state

13.12.2.78 OGRFieldDefnH OGR\_Fld\_Create ( const char \* *pszName*, OGRFieldType *eType* )

Create a new field definition.

This function is the same as the CPP method **OGRFieldDefn::OGRFieldDefn()** (p. ??).

## Parameters

<i>pszName</i>	the name of the new field definition.
<i>eType</i>	the type of the new field definition.

## Returns

handle to the new field definition.

13.12.2.79 void OGR\_Fld\_Destroy ( OGRFieldDefnH *hDefn* )

Destroy a field definition.

## Parameters

<i>hDefn</i>	handle to the field definition to destroy.
--------------	--

13.12.2.80 OGRJustification OGR\_Fld\_GetJustify ( OGRFieldDefnH *hDefn* )

Get the justification for this field.

This function is the same as the CPP method **OGRFieldDefn::GetJustify()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the field definition to get justification from.
--------------	---

## Returns

the justification.

**13.12.2.81** `const char* OGR_Fld_GetNameRef ( OGRFieldDefnH hDefn )`

Fetch name of this field.

This function is the same as the CPP method **OGRFieldDefn::GetNameRef()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the field definition.
--------------	---------------------------------

## Returns

the name of the field definition.

**13.12.2.82** `int OGR_Fld_GetPrecision ( OGRFieldDefnH hDefn )`

Get the formatting precision for this field. This should normally be zero for fields of types other than OFTReal.

This function is the same as the CPP method **OGRFieldDefn::GetPrecision()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the field definition to get precision from.
--------------	---

## Returns

the precision.

**13.12.2.83** `OGRFieldType OGR_Fld_GetType ( OGRFieldDefnH hDefn )`

Fetch type of this field.

This function is the same as the CPP method **OGRFieldDefn::GetType()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the field definition to get type from.
--------------	--

## Returns

field type.

**13.12.2.84** `int OGR_Fld_GetWidth ( OGRFieldDefnH hDefn )`

Get the formatting width for this field.

This function is the same as the CPP method **OGRFieldDefn::GetWidth()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the field definition to get width from.
--------------	---

## Returns

the width, zero means no specified width.

13.12.2.85 int OGR\_Fld\_IsIgnored ( OGRFieldDefnH *hDefn* )

Return whether this field should be omitted when fetching features.

This method is the same as the C++ method **OGRFieldDefn::IsIgnored()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the field definition
--------------	--------------------------------

## Returns

ignore state

13.12.2.86 void OGR\_Fld\_Set ( OGRFieldDefnH *hDefn*, const char \* *pszNameIn*, OGRFieldType *eTypeIn*, int *nWidthIn*, int *nPrecisionIn*, OGRJustification *eJustifyIn* )

Set defining parameters for a field in one call.

This function is the same as the CPP method **OGRFieldDefn::Set()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the field definition to set to.
<i>pszNameIn</i>	the new name to assign.
<i>eTypeIn</i>	the new type (one of the OFT values like OFTInteger).
<i>nWidthIn</i>	the preferred formatting width. Defaults to zero indicating undefined.
<i>nPrecisionIn</i>	number of decimals places for formatting, defaults to zero indicating undefined.
<i>eJustifyIn</i>	the formatting justification (OJLeft or OJRight), defaults to OJUndefined.

13.12.2.87 void OGR\_Fld\_SetIgnored ( OGRFieldDefnH *hDefn*, int *ignore* )

Set whether this field should be omitted when fetching features.

This method is the same as the C function **OGRFieldDefn::SetIgnored()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the field definition
<i>ignore</i>	ignore state

13.12.2.88 void OGR\_Fld\_SetJustify ( OGRFieldDefnH *hDefn*, OGRJustification *eJustify* )

Set the justification for this field.

This function is the same as the CPP method **OGRFieldDefn::SetJustify()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the field definition to set justification to.
<i>eJustify</i>	the new justification.

13.12.2.89 void OGR\_Fld\_SetName ( OGRFieldDefnH *hDefn*, const char \* *pszName* )

Reset the name of this field.

This function is the same as the CPP method **OGRFieldDefn::SetName()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the field definition to apply the new name to.
<i>pszName</i>	the new name to apply.

13.12.2.90 void OGR\_Fld\_SetPrecision ( OGRFieldDefnH *hDefn*, int *nPrecision* )

Set the formatting precision for this field in characters.

This should normally be zero for fields of types other than OFTReal.

This function is the same as the CPP method **OGRFieldDefn::SetPrecision()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the field definition to set precision to.
<i>nPrecision</i>	the new precision.

13.12.2.91 void OGR\_Fld\_SetType ( OGRFieldDefnH *hDefn*, OGRFieldType *eType* )

Set the type of this field. This should never be done to an **OGRFieldDefn** (p. ??) that is already part of an **OGR-FeatureDefn** (p. ??).

This function is the same as the CPP method **OGRFieldDefn::SetType()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the field definition to set type to.
<i>eType</i>	the new field type.

13.12.2.92 void OGR\_Fld\_SetWidth ( OGRFieldDefnH *hDefn*, int *nNewWidth* )

Set the formatting width for this field in characters.

This function is the same as the CPP method **OGRFieldDefn::SetWidth()** (p. ??).

## Parameters

<i>hDefn</i>	handle to the field definition to set width to.
<i>nNewWidth</i>	the new width.

13.12.2.93 OGRErr OGR\_G\_AddGeometry ( OGRGeometryH *hGeom*, OGRGeometryH *hNewSubGeom* )

Add a geometry to a geometry container.

Some subclasses of **OGRGeometryCollection** (p. ??) restrict the types of geometry that can be added, and may return an error. The passed geometry is cloned to make an internal copy.

There is no SFCOM analog to this method.

This function is the same as the CPP method **OGRGeometryCollection::addGeometry** (p. ??).

For a polygon, hNewSubGeom must be a linearring. If the polygon is empty, the first added subgeometry will be the exterior ring. The next ones will be the interior rings.

#### Parameters

<i>hGeom</i>	existing geometry container.
<i>hNewSubGeom</i>	geometry to add to the container.

#### Returns

OGRERR\_NONE if successful, or OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE if the geometry type is illegal for the type of existing geometry.

References wkbGeometryCollection, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, and wkbPolygon.

#### 13.12.2.94 OGRErr OGR\_G\_AddGeometryDirectly ( OGRGeometryH *hGeom*, OGRGeometryH *hNewSubGeom* )

Add a geometry directly to an existing geometry container.

Some subclasses of **OGRGeometryCollection** (p. ??) restrict the types of geometry that can be added, and may return an error. Ownership of the passed geometry is taken by the container rather than cloning as addGeometry() does.

This function is the same as the CPP method **OGRGeometryCollection::addGeometryDirectly** (p. ??).

There is no SFCOM analog to this method.

For a polygon, hNewSubGeom must be a linearring. If the polygon is empty, the first added subgeometry will be the exterior ring. The next ones will be the interior rings.

#### Parameters

<i>hGeom</i>	existing geometry.
<i>hNewSubGeom</i>	geometry to add to the existing geometry.

#### Returns

OGRERR\_NONE if successful, or OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE if the geometry type is illegal for the type of geometry container.

References wkbGeometryCollection, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, and wkbPolygon.

#### 13.12.2.95 void OGR\_G\_AddPoint ( OGRGeometryH *hGeom*, double *dfX*, double *dfY*, double *dfZ* )

Add a point to a geometry (line string or point).

The vertex count of the line string is increased by one, and assigned from the passed location value.

#### Parameters

<i>hGeom</i>	handle to the geometry to add a point to.
<i>dfX</i>	x coordinate of point to add.
<i>dfY</i>	y coordinate of point to add.
<i>dfZ</i>	z coordinate of point to add.



References CPLError(), wkbLineString, and wkbPoint.

13.12.2.96 void OGR\_G\_AddPoint\_2D ( OGRGeometryH *hGeom*, double *dfX*, double *dfY* )

Add a point to a geometry (line string or point).

The vertex count of the line string is increased by one, and assigned from the passed location value.

#### Parameters

<i>hGeom</i>	handle to the geometry to add a point to.
<i>dfX</i>	x coordinate of point to add.
<i>dfY</i>	y coordinate of point to add.

References CPLError(), wkbLineString, and wkbPoint.

13.12.2.97 OGRGeometryH OGR\_G\_ApproximateArcAngles ( double *dfCenterX*, double *dfCenterY*, double *dfZ*, double *dfPrimaryRadius*, double *dfSecondaryRadius*, double *dfRotation*, double *dfStartAngle*, double *dfEndAngle*, double *dfMaxAngleStepSizeDegrees* )

Stroke arc to linestring.

Stroke an arc of a circle to a linestring based on a center point, radius, start angle and end angle, all angles in degrees.

If the *dfMaxAngleStepSizeDegrees* is zero, then a default value will be used. This is currently 4 degrees unless the user has overridden the value with the OGR\_ARC\_STEPSIZE configuration variable.

#### See Also

**CPLSetConfigOption()** (p. ??)

#### Parameters

<i>dfCenterX</i>	center X
<i>dfCenterY</i>	center Y
<i>dfZ</i>	center Z
<i>dfPrimaryRadius</i>	X radius of ellipse.
<i>dfSecondaryRadius</i>	Y radius of ellipse.
<i>dfRotation</i>	rotation of the ellipse clockwise.
<i>dfStartAngle</i>	angle to first point on arc (clockwise of X-positive)
<i>dfEndAngle</i>	angle to last point on arc (clockwise of X-positive)
<i>dfMaxAngleStepSizeDegrees</i>	the largest step in degrees along the arc, zero to use the default setting.

#### Returns

**OGRLineString** (p. ??) geometry representing an approximation of the arc.

#### Since

OGR 1.8.0

References OGRGeometryFactory::approximateArcAngles().

### 13.12.2.98 double OGR\_G\_Area ( OGRGeometryH *hGeom* )

Compute geometry area.

Computes the area for an **OGRLinearRing** (p. ??), **OGRPolygon** (p. ??) or **OGRMultiPolygon** (p. ??). Undefined for all other geometry types (returns zero).

This function utilizes the C++ `get_Area()` methods such as **OGRPolygon::get\_Area()** (p. ??).

#### Parameters

<i>hGeom</i>	the geometry to operate on.
--------------	-----------------------------

#### Returns

the area or 0.0 for unsupported geometry types.

#### Since

OGR 1.8.0

References `CPLError()`, `wkbGeometryCollection`, `wkbLinearRing`, `wkbLineString`, `wkbMultiPolygon`, and `wkbPolygon`.

Referenced by `OGRFeature::GetFieldAsDouble()`, `OGRFeature::GetFieldAsInteger()`, `OGRFeature::GetFieldAsString()`, `OGRFeature::IsFieldSet()`, and `OGR_G_GetArea()`.

### 13.12.2.99 void OGR\_G\_AssignSpatialReference ( OGRGeometryH *hGeom*, OGRSpatialReferenceH *hSRS* )

Assign spatial reference to this object.

Any existing spatial reference is replaced, but under no circumstances does this result in the object being re-projected. It is just changing the interpretation of the existing geometry. Note that assigning a spatial reference increments the reference count on the **OGRSpatialReference** (p. ??), but does not copy it.

This is similar to the SFCOM `IGeometry::put_SpatialReference()` method.

This function is the same as the CPP method **OGRGeometry::assignSpatialReference** (p. ??).

#### Parameters

<i>hGeom</i>	handle on the geometry to apply the new spatial reference system.
<i>hSRS</i>	handle on the new spatial reference system to apply.

### 13.12.2.100 OGRGeometryH OGR\_G\_Boundary ( OGRGeometryH *hTarget* )

Compute boundary.

A new geometry object is created and returned containing the boundary of the geometry on which the method is invoked.

This function is the same as the C++ method **OGR\_G\_Boundary()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a `CPL_NotSupported` error.

#### Parameters

<i>hTarget</i>	The Geometry to calculate the boundary of.
----------------	--

**Returns**

a handle to a newly allocated geometry now owned by the caller, or NULL on failure.

**Since**

OGR 1.8.0

**13.12.2.101 OGRGeometryH OGR\_G\_Buffer ( OGRGeometryH hTarget, double dfDist, int nQuadSegs )**

Compute buffer of geometry.

Builds a new geometry containing the buffer region around the geometry on which it is invoked. The buffer is a polygon containing the region within the buffer distance of the original geometry.

Some buffer sections are properly described as curves, but are converted to approximate polygons. The nQuadSegs parameter can be used to control how many segments should be used to define a 90 degree curve - a quadrant of a circle. A value of 30 is a reasonable default. Large values result in large numbers of vertices in the resulting buffer geometry while small numbers reduce the accuracy of the result.

This function is the same as the C++ method **OGRGeometry::Buffer()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

**Parameters**

<i>hTarget</i>	the geometry.
<i>dfDist</i>	the buffer distance to be applied.
<i>nQuadSegs</i>	the number of segments used to approximate a 90 degree (quadrant) of curvature.

**Returns**

the newly created geometry, or NULL if an error occurs.

**13.12.2.102 int OGR\_G\_Centroid ( OGRGeometryH hGeom, OGRGeometryH hCentroidPoint )**

Compute the geometry centroid.

The centroid location is applied to the passed in **OGRPoint** (p. ??) object. The centroid is not necessarily within the geometry.

This method relates to the SFCOM ISurface::get\_Centroid() method however the current implementation based on GEOS can operate on other geometry types such as multipoint, linestring, geometrycollection such as multipolygons. OGC SF SQL 1.1 defines the operation for surfaces (polygons). SQL/MM-Part 3 defines the operation for surfaces and multisurfaces (multipolygons).

This function is the same as the C++ method **OGRGeometry::Centroid()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

**Returns**

OGRERR\_NONE on success or OGRERR\_FAILURE on error.

References **OGRGeometry::Centroid()**, **CPLError()**, **OGRPoint::getGeometryType()**, and **wkbPoint**.

### 13.12.2.103 OGRGeometryH OGR\_G\_Clone ( OGRGeometryH *hGeom* )

Make a copy of this object.

This function relates to the SFCOM IGeometry::clone() method.

This function is the same as the CPP method **OGRGeometry::clone()** (p. ??).

#### Parameters

<i>hGeom</i>	handle on the geometry to clone from.
--------------	---------------------------------------

#### Returns

an handle on the copy of the geometry with the spatial reference system as the original.

### 13.12.2.104 void OGR\_G\_CloseRings ( OGRGeometryH *hGeom* )

Force rings to be closed.

If this geometry, or any contained geometries has polygon rings that are not closed, they will be closed by adding the starting point at the end.

#### Parameters

<i>hGeom</i>	handle to the geometry.
--------------	-------------------------

### 13.12.2.105 int OGR\_G\_Contains ( OGRGeometryH *hThis*, OGRGeometryH *hOther* )

Test for containment.

Tests if this geometry contains the other geometry.

This function is the same as the C++ method **OGRGeometry::Contains()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

#### Parameters

<i>hThis</i>	the geometry to compare.
<i>hOther</i>	the other geometry to compare.

#### Returns

TRUE if *hThis* contains *hOther* geometry, otherwise FALSE.

### 13.12.2.106 OGRGeometryH OGR\_G\_ConvexHull ( OGRGeometryH *hTarget* )

Compute convex hull.

A new geometry object is created and returned containing the convex hull of the geometry on which the method is invoked.

This function is the same as the C++ method **OGRGeometry::ConvexHull()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>hTarget</i>	The Geometry to calculate the convex hull of.
----------------	---

## Returns

a handle to a newly allocated geometry now owned by the caller, or NULL on failure.

13.12.2.107 OGRGeometryH OGR\_G\_CreateFromGML ( const char \* *pszGML* )

Create geometry from GML.

This method translates a fragment of GML containing only the geometry portion into a corresponding **OGR-Geometry** (p. ??). There are many limitations on the forms of GML geometries supported by this parser, but they are too numerous to list here.

The following GML2 elements are parsed : Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, MultiGeometry.

(OGR >= 1.8.0) The following GML3 elements are parsed : Surface, MultiSurface, PolygonPatch, Triangle, Rectangle, Curve, MultiCurve, CompositeCurve, LineStringSegment, Arc, Circle, CompositeSurface, OrientableSurface, Solid, Tin, TriangulatedSurface.

Arc and Circle elements are stroked to linestring, by using a 4 degrees step, unless the user has overridden the value with the OGR\_ARC\_STEPSIZE configuration variable.

The C++ method **OGRGeometryFactory::createFromGML()** (p. ??) is the same as this function.

## Parameters

<i>pszGML</i>	The GML fragment for the geometry.
---------------	------------------------------------

## Returns

a geometry on succes, or NULL on error.

References CPLDestroyXMLNode(), CPLError(), CPLGetConfigOption(), CPLParseXMLString(), and CSLTestBoolean().

Referenced by OGRGeometryFactory::createFromGML().

13.12.2.108 OGRErr OGR\_G\_CreateFromWkb ( unsigned char \* *pabyData*, OGRSpatialReferenceH *hSRS*, OGRGeometryH \* *phGeometry*, int *nBytes* )

Create a geometry object of the appropriate type from it's well known binary representation.

Note that if nBytes is passed as zero, no checking can be done on whether the pabyData is sufficient. This can result in a crash if the input data is corrupt. This function returns no indication of the number of bytes from the data source actually used to represent the returned geometry object. Use **OGR\_G\_WkbSize()** (p. ??) on the returned geometry to establish the number of bytes it required in WKB format.

The **OGRGeometryFactory::createFromWkb()** (p. ??) CPP method is the same as this function.

## Parameters

<i>pabyData</i>	pointer to the input BLOB data.
<i>hSRS</i>	handle to the spatial reference to be assigned to the created geometry object. This may be NULL.

<i>phGeometry</i>	the newly created geometry object will be assigned to the indicated handle on return. This will be NULL in case of failure. If not NULL, *phGeometry should be freed with <b>OGR_G_DestroyGeometry()</b> (p. ??) after use.
<i>nBytes</i>	the number of bytes of data available in pabyData, or -1 if it is not known, but assumed to be sufficient.

#### Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

References OGRGeometryFactory::createFromWkb().

13.12.2.109 OGRErr OGR\_G\_CreateFromWkt ( char \*\* *ppszData*, OGRSpatialReferenceH *hSRS*, OGRGeometryH \* *phGeometry* )

Create a geometry object of the appropriate type from it's well known text representation.

The **OGRGeometryFactory::createFromWkt** (p. ??) CPP method is the same as this function.

#### Parameters

<i>ppszData</i>	input zero terminated string containing well known text representation of the geometry to be created. The pointer is updated to point just beyond that last character consumed.
<i>hSRS</i>	handle to the spatial reference to be assigned to the created geometry object. This may be NULL.
<i>phGeometry</i>	the newly created geometry object will be assigned to the indicated handle on return. This will be NULL if the method fails. If not NULL, *phGeometry should be freed with <b>OGR_G_DestroyGeometry()</b> (p. ??) after use.

#### Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

References OGRGeometryFactory::createFromWkt().

13.12.2.110 OGRGeometryH OGR\_G\_CreateGeometry ( OGRwkbGeometryType *eGeometryType* )

Create an empty geometry of desired type.

This is equivalent to allocating the desired geometry with new, but the allocation is guaranteed to take place in the context of the GDAL/OGR heap.

This function is the same as the CPP method **OGRGeometryFactory::createGeometry** (p. ??).

#### Parameters

<i>eGeometryType</i>	the type code of the geometry to be created.
----------------------	--

#### Returns

handle to the newly create geometry or NULL on failure. Should be freed with **OGR\_G\_DestroyGeometry()** (p. ??) after use.

References OGRGeometryFactory::createGeometry().

13.12.2.111 `int OGR_G_Crosses ( OGRGeometryH hThis, OGRGeometryH hOther )`

Test for crossing.

Tests if this geometry and the other geometry are crossing.

This function is the same as the C++ method **OGRGeometry::Crosses()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

#### Parameters

<i>hThis</i>	the geometry to compare.
<i>hOther</i>	the other geometry to compare.

#### Returns

TRUE if they are crossing, otherwise FALSE.

13.12.2.112 `void OGR_G_DestroyGeometry ( OGRGeometryH hGeom )`

Destroy geometry object.

Equivalent to invoking delete on a geometry, but it guaranteed to take place within the context of the GDAL/OGR heap.

This function is the same as the CPP method **OGRGeometryFactory::destroyGeometry** (p. ??).

#### Parameters

<i>hGeom</i>	handle to the geometry to delete.
--------------	-----------------------------------

References **OGRGeometryFactory::destroyGeometry()**.

13.12.2.113 `OGRGeometryH OGR_G_Difference ( OGRGeometryH hThis, OGRGeometryH hOther )`

Compute difference.

Generates a new geometry which is the region of this geometry with the region of the other geometry removed.

This function is the same as the C++ method **OGRGeometry::Difference()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

#### Parameters

<i>hThis</i>	the geometry.
<i>hOther</i>	the other geometry.

#### Returns

a new geometry representing the difference or NULL if the difference is empty or an error occurs.

13.12.2.114 `int OGR_G_Disjoint ( OGRGeometryH hThis, OGRGeometryH hOther )`

Test for disjointness.

Tests if this geometry and the other geometry are disjoint.

This function is the same as the C++ method **OGRGeometry::Disjoint()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a `CPLE_NotSupported` error.



## Parameters

<i>hThis</i>	the geometry to compare.
<i>hOther</i>	the other geometry to compare.

## Returns

TRUE if they are disjoint, otherwise FALSE.

13.12.2.115 `double OGR_G_Distance ( OGRGeometryH hFirst, OGRGeometryH hOther )`

Compute distance between two geometries.

Returns the shortest distance between the two geometries.

This function is the same as the C++ method **OGRGeometry::Distance()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>hFirst</i>	the first geometry to compare against.
<i>hOther</i>	the other geometry to compare against.

## Returns

the distance between the geometries or -1 if an error occurs.

13.12.2.116 `void OGR_G_DumpReadable ( OGRGeometryH hGeom, FILE * fp, const char * pszPrefix )`

Dump geometry in well known text format to indicated output file.

This method is the same as the CPP method **OGRGeometry::dumpReadable** (p. ??).

## Parameters

<i>hGeom</i>	handle on the geometry to dump.
<i>fp</i>	the text file to write the geometry to.
<i>pszPrefix</i>	the prefix to put on each line of output.

13.12.2.117 `void OGR_G_Empty ( OGRGeometryH hGeom )`

Clear geometry information. This restores the geometry to it's initial state after construction, and before assignment of actual geometry.

This function relates to the SFCOM IGeometry::Empty() method.

This function is the same as the CPP method **OGRGeometry::empty()** (p. ??).

## Parameters

<i>hGeom</i>	handle on the geometry to empty.
--------------	----------------------------------

13.12.2.118 `int OGR_G_Equals ( OGRGeometryH hGeom, OGRGeometryH hOther )`

Returns TRUE if two geometries are equivalent.

This function is the same as the CPP method **OGRGeometry::Equals()** (p. ??) method.

## Parameters

<i>hGeom</i>	handle on the first geometry.
<i>hOther</i>	handle on the other geometry to test against.

## Returns

TRUE if equivalent or FALSE otherwise.

References CPLError().

### 13.12.2.119 char\* OGR\_G\_ExportToGML ( OGRGeometryH *hGeometry* )

Convert a geometry into GML format.

The GML geometry is expressed directly in terms of GML basic data types assuming the this is available in the gml namespace. The returned string should be freed with CPLFree() when no longer required.

This method is the same as the C++ method **OGRGeometry::exportToGML()** (p. ??).

## Parameters

<i>hGeometry</i>	handle to the geometry.
------------------	-------------------------

## Returns

A GML fragment or NULL in case of error.

References OGR\_G\_ExportToGMLEx().

### 13.12.2.120 char\* OGR\_G\_ExportToGMLEx ( OGRGeometryH *hGeometry*, char \*\* *papszOptions* )

Convert a geometry into GML format.

The GML geometry is expressed directly in terms of GML basic data types assuming the this is available in the gml namespace. The returned string should be freed with CPLFree() when no longer required.

The supported options in OGR 1.8.0 are :

- FORMAT=GML3. Otherwise it will default to GML 2.1.2 output.
- GML3\_LINESTRING\_ELEMENT=curve. (Only valid for FORMAT=GML3) To use gml:Curve element for linestrings. Otherwise gml:LineString will be used .
- GML3\_LONGSRs=YES/NO. (Only valid for FORMAT=GML3) Default to YES. If YES, SRS with EPSG authority will be written with the "urn:ogc:def:crs:EPSG:." prefix. In the case, if the SRS is a geographic SRS without explicit AXIS order, but that the same SRS authority code imported with ImportFromEPSGA() should be treated as lat/long, then the function will take care of coordinate order swapping. If set to NO, SRS with EPSG authority will be written with the "EPSG:" prefix, even if they are in lat/long order.
- GMLID=astring. If specified, a gml:id attribute will be written in the top-level geometry element with the provided value. Required for GML 3.2 compatibility.

This method is the same as the C++ method **OGRGeometry::exportToGML()** (p. ??).

## Parameters

<i>hGeometry</i>	handle to the geometry.
<i>papszOptions</i>	NULL-terminated list of options.

**Returns**

A GML fragment or NULL in case of error.

**Since**

OGR 1.8.0

References CPLMalloc(), CPLStrdup(), and CSLTestBoolean().

Referenced by OGRGeometry::exportToGML(), and OGR\_G\_ExportToGML().

**13.12.2.121 char\* OGR\_G\_ExportToJson ( OGRGeometryH *hGeometry* )**

Convert a geometry into GeoJSON format.

The returned string should be freed with CPLFree() when no longer required.

This method is the same as the C++ method **OGRGeometry::exportToJson()** (p. ??).

**Parameters**

<i>hGeometry</i>	handle to the geometry.
------------------	-------------------------

**Returns**

A GeoJSON fragment or NULL in case of error.

References OGR\_G\_ExportToJsonEx().

Referenced by OGRGeometry::exportToJson().

**13.12.2.122 char\* OGR\_G\_ExportToJsonEx ( OGRGeometryH *hGeometry*, char \*\* *papszOptions* )**

Convert a geometry into GeoJSON format.

The returned string should be freed with CPLFree() when no longer required.

This method is the same as the C++ method **OGRGeometry::exportToJson()** (p. ??).

**Parameters**

<i>hGeometry</i>	handle to the geometry.
<i>papszOptions</i>	a null terminated list of options. For now, only COORDINATE_PRECISION=int_number where int_number is the maximum number of figures after decimal separator to write in coordinates.

**Returns**

A GeoJSON fragment or NULL in case of error.

**Since**

OGR 1.9.0

References CPLStrdup().

Referenced by OGR\_G\_ExportToJson().

13.12.2.123 `char* OGR_G_ExportToKML ( OGRGeometryH hGeometry, const char * pszAltitudeMode )`

Convert a geometry into KML format.

The returned string should be freed with `CPLFree()` when no longer required.

This method is the same as the C++ method `OGRGeometry::exportToKML()` (p. ??).

#### Parameters

<i>hGeometry</i>	handle to the geometry.
<i>pszAltitudeMode</i>	value to write in altitudeMode element, or NULL.

#### Returns

A KML fragment or NULL in case of error.

References `CPLMalloc()`, and `CPLStrdup()`.

Referenced by `OGRGeometry::exportToKML()`.

13.12.2.124 `OGRERR OGR_G_ExportToWkb ( OGRGeometryH hGeom, OGRwkbByteOrder eOrder, unsigned char * pabyDstBuffer )`

Convert a geometry into well known binary format.

This function relates to the `SFCOM IWks::ExportToWKB()` method.

This function is the same as the CPP method `OGRGeometry::exportToWkb()` (p. ??).

#### Parameters

<i>hGeom</i>	handle on the geometry to convert to a well know binary data from.
<i>eOrder</i>	One of <code>wkbXDR</code> or <code>wkbNDR</code> indicating MSB or LSB byte order respectively.
<i>pabyDstBuffer</i>	a buffer into which the binary representation is written. This buffer must be at least <code>OGR_G-_WkbSize()</code> (p. ??) byte in size.

#### Returns

Currently `OGRERR_NONE` is always returned.

13.12.2.125 `OGRERR OGR_G_ExportToWkt ( OGRGeometryH hGeom, char ** ppszSrcText )`

Convert a geometry into well known text format.

This function relates to the `SFCOM IWks::ExportToWKT()` method.

This function is the same as the CPP method `OGRGeometry::exportToWkt()` (p. ??).

#### Parameters

<i>hGeom</i>	handle on the geometry to convert to a text format from.
<i>ppszSrcText</i>	a text buffer is allocated by the program, and assigned to the passed pointer. After use, <code>*ppszDstText</code> should be freed with <code>OGRFree()</code> .

#### Returns

Currently `OGRERR_NONE` is always returned.

13.12.2.126 void OGR\_G\_FlattenTo2D ( OGRGeometryH *hGeom* )

Convert geometry to strictly 2D. In a sense this converts all Z coordinates to 0.0.

This function is the same as the CPP method **OGRGeometry::flattenTo2D()** (p. ??).

## Parameters

<i>hGeom</i>	handle on the geometry to convert.
--------------	------------------------------------

**13.12.2.127 OGRGeometryH OGR\_G\_ForceToLineString ( OGRGeometryH *hGeom* )**

Convert to line string.

This function is the same as the C++ method **OGRGeometryFactory::forceToLineString()** (p. ??).

## Parameters

<i>hGeom</i>	handle to the geometry to convert (ownership surrendered).
--------------	--

## Returns

the converted geometry (ownership to caller).

## Since

GDAL/OGR 1.10.0

References OGRGeometryFactory::forceToLineString().

**13.12.2.128 OGRGeometryH OGR\_G\_ForceToMultiLineString ( OGRGeometryH *hGeom* )**

Convert to multilinestring.

This function is the same as the C++ method **OGRGeometryFactory::forceToMultiLineString()** (p. ??).

## Parameters

<i>hGeom</i>	handle to the geometry to convert (ownership surrendered).
--------------	--

## Returns

the converted geometry (ownership to caller).

## Since

GDAL/OGR 1.8.0

References OGRGeometryFactory::forceToMultiLineString().

**13.12.2.129 OGRGeometryH OGR\_G\_ForceToMultiPoint ( OGRGeometryH *hGeom* )**

Convert to multipoint.

This function is the same as the C++ method **OGRGeometryFactory::forceToMultiPoint()** (p. ??).

## Parameters

<i>hGeom</i>	handle to the geometry to convert (ownership surrendered).
--------------	--

## Returns

the converted geometry (ownership to caller).

## Since

GDAL/OGR 1.8.0

References OGRGeometryFactory::forceToMultiPoint().

**13.12.2.130 OGRGeometryH OGR\_G\_ForceToMultiPolygon ( OGRGeometryH *hGeom* )**

Convert to multipolygon.

This function is the same as the C++ method **OGRGeometryFactory::forceToMultiPolygon()** (p. ??).

**Parameters**

<i>hGeom</i>	handle to the geometry to convert (ownership surrendered).
--------------	--

**Returns**

the converted geometry (ownership to caller).

**Since**

GDAL/OGR 1.8.0

References **OGRGeometryFactory::forceToMultiPolygon()**.

**13.12.2.131 OGRGeometryH OGR\_G\_ForceToPolygon ( OGRGeometryH *hGeom* )**

Convert to polygon.

This function is the same as the C++ method **OGRGeometryFactory::forceToPolygon()** (p. ??).

**Parameters**

<i>hGeom</i>	handle to the geometry to convert (ownership surrendered).
--------------	--

**Returns**

the converted geometry (ownership to caller).

**Since**

GDAL/OGR 1.8.0

References **OGRGeometryFactory::forceToPolygon()**.

**13.12.2.132 double OGR\_G\_GetArea ( OGRGeometryH *hGeom* )**

Compute geometry area (deprecated)

**Deprecated****See Also**

**OGR\_G\_Area()** (p. ??)

References **OGR\_G\_Area()**.

**13.12.2.133 OGRGeometryH OGR\_G\_GetBoundary ( OGRGeometryH *hTarget* )**

Compute boundary (deprecated)

**Deprecated****See Also**

**OGR\_G\_Boundary()** (p. ??)

### 13.12.2.134 int OGR\_G\_GetCoordinateDimension ( OGRGeometryH *hGeom* )

Get the dimension of the coordinates in this geometry.

This function corresponds to the SFCOM IGeometry::GetDimension() method.

This function is the same as the CPP method **OGRGeometry::getCoordinateDimension()** (p. ??).

#### Parameters

<i>hGeom</i>	handle on the geometry to get the dimension of the coordinates from.
--------------	--

#### Returns

in practice this will return 2 or 3. It can also return 0 in the case of an empty point.

### 13.12.2.135 int OGR\_G\_GetDimension ( OGRGeometryH *hGeom* )

Get the dimension of this geometry.

This function corresponds to the SFCOM IGeometry::GetDimension() method. It indicates the dimension of the geometry, but does not indicate the dimension of the underlying space (as indicated by **OGR\_G\_GetCoordinateDimension()** (p. ??) function).

This function is the same as the CPP method **OGRGeometry::getDimension()** (p. ??).

#### Parameters

<i>hGeom</i>	handle on the geometry to get the dimension from.
--------------	---

#### Returns

0 for points, 1 for lines and 2 for surfaces.

### 13.12.2.136 void OGR\_G\_GetEnvelope ( OGRGeometryH *hGeom*, OGREnvelope \* *psEnvelope* )

Computes and returns the bounding envelope for this geometry in the passed *psEnvelope* structure.

This function is the same as the CPP method **OGRGeometry::getEnvelope()** (p. ??).

#### Parameters

<i>hGeom</i>	handle of the geometry to get envelope from.
<i>psEnvelope</i>	the structure in which to place the results.

### 13.12.2.137 void OGR\_G\_GetEnvelope3D ( OGRGeometryH *hGeom*, OGREnvelope3D \* *psEnvelope* )

Computes and returns the bounding envelope (3D) for this geometry in the passed *psEnvelope* structure.

This function is the same as the CPP method **OGRGeometry::getEnvelope()** (p. ??).

#### Parameters

<i>hGeom</i>	handle of the geometry to get envelope from.
<i>psEnvelope</i>	the structure in which to place the results.

#### Since

OGR 1.9.0



13.12.2.138 int OGR\_G\_GetGeometryCount ( OGRGeometryH *hGeom* )

Fetch the number of elements in a geometry or number of geometries in container.

Only geometries of type wkbPolygon[25D], wkbMultiPoint[25D], wkbMultiLineString[25D], wkbMultiPolygon[25D] or wkbGeometryCollection[25D] may return a valid value. Other geometry types will silently return 0.

For a polygon, the returned number is the number of rings (exterior ring + interior rings).

## Parameters

<i>hGeom</i>	single geometry or geometry container from which to get the number of elements.
--------------	---

## Returns

the number of elements.

References wkbGeometryCollection, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, and wkbPolygon.

13.12.2.139 const char\* OGR\_G\_GetGeometryName ( OGRGeometryH *hGeom* )

Fetch WKT name for geometry type.

There is no SFCOM analog to this function.

This function is the same as the CPP method **OGRGeometry::getGeometryName()** (p. ??).

## Parameters

<i>hGeom</i>	handle on the geometry to get name from.
--------------	--

## Returns

name used for this geometry type in well known text format.

13.12.2.140 OGRGeometryH OGR\_G\_GetGeometryRef ( OGRGeometryH *hGeom*, int *iSubGeom* )

Fetch geometry from a geometry container.

This function returns an handle to a geometry within the container. The returned geometry remains owned by the container, and should not be modified. The handle is only valid untill the next change to the geometry container. Use **OGR\_G\_Clone()** (p. ??) to make a copy.

This function relates to the SFCOM IGeometryCollection::get\_Geometry() method.

This function is the same as the CPP method **OGRGeometryCollection::getGeometryRef()** (p. ??).

For a polygon, OGR\_G\_GetGeometryRef(*iSubGeom*) returns the exterior ring if *iSubGeom* == 0, and the interior rings for *iSubGeom* > 0.

## Parameters

<i>hGeom</i>	handle to the geometry container from which to get a geometry from.
<i>iSubGeom</i>	the index of the geometry to fetch, between 0 and getNumGeometries() - 1.

## Returns

handle to the requested geometry.

References CPLError(), wkbGeometryCollection, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, and wkbPolygon.

### 13.12.2.141 **OGRwkbGeometryType OGR\_G\_GetGeometryType ( OGRGeometryH *hGeom* )**

Fetch geometry type.

Note that the geometry type may include the 2.5D flag. To get a 2D flattened version of the geometry type apply the `wkbFlatten()` macro to the return result.

This function is the same as the CPP method **OGRGeometry::getGeometryType()** (p. ??).

#### Parameters

<i>hGeom</i>	handle on the geometry to get type from.
--------------	--

#### Returns

the geometry type code.

References `wkbUnknown`.

### 13.12.2.142 **void OGR\_G\_GetPoint ( OGRGeometryH *hGeom*, int *i*, double \* *pdfX*, double \* *pdfY*, double \* *pdfZ* )**

Fetch a point in line string or a point geometry.

#### Parameters

<i>hGeom</i>	handle to the geometry from which to get the coordinates.
<i>i</i>	the vertex to fetch, from 0 to <code>getNumPoints()-1</code> , zero for a point.
<i>pdfX</i>	value of x coordinate.
<i>pdfY</i>	value of y coordinate.
<i>pdfZ</i>	value of z coordinate.

References `CPLError()`, `OGRLineString::getNumPoints()`, `OGRLineString::getX()`, `OGRLineString::getY()`, `OGRLineString::getZ()`, `wkbLineString`, and `wkbPoint`.

### 13.12.2.143 **int OGR\_G\_GetPointCount ( OGRGeometryH *hGeom* )**

Fetch number of points from a geometry.

Only `wkbPoint[25D]` or `wkbLineString[25D]` may return a valid value. Other geometry types will silently return 0.

#### Parameters

<i>hGeom</i>	handle to the geometry from which to get the number of points.
--------------	--

#### Returns

the number of points.

References `OGRLineString::getNumPoints()`, `wkbLineString`, and `wkbPoint`.

### 13.12.2.144 **int OGR\_G\_GetPoints ( OGRGeometryH *hGeom*, void \* *pabyX*, int *nXStride*, void \* *pabyY*, int *nYStride*, void \* *pabyZ*, int *nZStride* )**

Returns all points of line string.

This method copies all points into user arrays. The user provides the stride between 2 consecutives elements of the array.

On some CPU architectures, care must be taken so that the arrays are properly aligned.

## Parameters

<i>hGeom</i>	handle to the geometry from which to get the coordinates.
<i>pabyX</i>	a buffer of at least (sizeof(double) * nXStride * nPointCount) bytes, may be NULL.
<i>nXStride</i>	the number of bytes between 2 elements of pabyX.
<i>pabyY</i>	a buffer of at least (sizeof(double) * nYStride * nPointCount) bytes, may be NULL.
<i>nYStride</i>	the number of bytes between 2 elements of pabyY.
<i>pabyZ</i>	a buffer of at last size (sizeof(double) * nZStride * nPointCount) bytes, may be NULL.
<i>nZStride</i>	the number of bytes between 2 elements of pabyZ.

## Returns

the number of points

## Since

OGR 1.9.0

References CPLError(), OGRLineString::getNumPoints(), OGRLineString::getPoints(), wkbLineString, and wkbPoint.

#### 13.12.2.145 OGRSpatialReferenceH OGR\_G\_GetSpatialReference ( OGRGeometryH *hGeom* )

Returns spatial reference system for geometry.

This function relates to the SFCOM IGeometry::get\_SpatialReference() method.

This function is the same as the CPP method **OGRGeometry::getSpatialReference()** (p. ??).

## Parameters

<i>hGeom</i>	handle on the geometry to get spatial reference from.
--------------	---

## Returns

a reference to the spatial reference geometry.

#### 13.12.2.146 double OGR\_G\_GetX ( OGRGeometryH *hGeom*, int *i* )

Fetch the x coordinate of a point from a geometry.

## Parameters

<i>hGeom</i>	handle to the geometry from which to get the x coordinate.
<i>i</i>	point to get the x coordinate.

## Returns

the X coordinate of this point.

References CPLError(), OGRLineString::getNumPoints(), OGRLineString::getX(), wkbLineString, and wkbPoint.

#### 13.12.2.147 double OGR\_G\_GetY ( OGRGeometryH *hGeom*, int *i* )

Fetch the x coordinate of a point from a geometry.

## Parameters

<i>hGeom</i>	handle to the geometry from which to get the y coordinate.
<i>i</i>	point to get the Y coordinate.

## Returns

the Y coordinate of this point.

References CPLError(), OGRLineString::getNumPoints(), OGRLineString::getY(), wkbLineString, and wkbPoint.

### 13.12.2.148 double OGR\_G\_GetZ ( OGRGeometryH *hGeom*, int *i* )

Fetch the z coordinate of a point from a geometry.

## Parameters

<i>hGeom</i>	handle to the geometry from which to get the Z coordinate.
<i>i</i>	point to get the Z coordinate.

## Returns

the Z coordinate of this point.

References CPLError(), OGRLineString::getNumPoints(), OGRLineString::getZ(), wkbLineString, and wkbPoint.

### 13.12.2.149 OGRErr OGR\_G\_ImportFromWkb ( OGRGeometryH *hGeom*, unsigned char \* *pabyData*, int *nSize* )

Assign geometry from well known binary data.

The object must have already been instantiated as the correct derived type of geometry object to match the binaries type.

This function relates to the SFCOM IWks::ImportFromWKB() method.

This function is the same as the CPP method **OGRGeometry::importFromWkb()** (p. ??).

## Parameters

<i>hGeom</i>	handle on the geometry to assign the well know binary data to.
<i>pabyData</i>	the binary input data.
<i>nSize</i>	the size of pabyData in bytes, or zero if not known.

## Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

### 13.12.2.150 OGRErr OGR\_G\_ImportFromWkt ( OGRGeometryH *hGeom*, char \*\* *ppsSrcText* )

Assign geometry from well known text data.

The object must have already been instantiated as the correct derived type of geometry object to match the text type.

This function relates to the SFCOM IWks::ImportFromWKT() method.

This function is the same as the CPP method **OGRGeometry::importFromWkt()** (p. ??).

## Parameters

<i>hGeom</i>	handle on the geometry to assign well know text data to.
<i>ppszSrcText</i>	pointer to a pointer to the source text. The pointer is updated to pointer after the consumed text.

## Returns

OGRERR\_NONE if all goes well, otherwise any of OGRERR\_NOT\_ENOUGH\_DATA, OGRERR\_UNSUPPORTED\_GEOMETRY\_TYPE, or OGRERR\_CORRUPT\_DATA may be returned.

13.12.2.151 OGRGeometryH OGR\_G\_Intersection ( OGRGeometryH *hThis*, OGRGeometryH *hOther* )

Compute intersection.

Generates a new geometry which is the region of intersection of the two geometries operated on. The **OGR\_G\_Intersection()** (p. ??) function can be used to test if two geometries intersect.

This function is the same as the C++ method **OGRGeometry::Intersection()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>hThis</i>	the geometry.
<i>hOther</i>	the other geometry.

## Returns

a new geometry representing the intersection or NULL if there is no intersection or an error occurs.

13.12.2.152 int OGR\_G\_Intersects ( OGRGeometryH *hGeom*, OGRGeometryH *hOtherGeom* )

Do these features intersect?

Currently this is not implemented in a rigorous fashion, and generally just tests whether the envelopes of the two features intersect. Eventually this will be made rigorous.

This function is the same as the CPP method **OGRGeometry::Intersects** (p. ??).

## Parameters

<i>hGeom</i>	handle on the first geometry.
<i>hOtherGeom</i>	handle on the other geometry to test against.

## Returns

TRUE if the geometries intersect, otherwise FALSE.

13.12.2.153 int OGR\_G\_IsEmpty ( OGRGeometryH *hGeom* )

Test if the geometry is empty.

This method is the same as the CPP method **OGRGeometry::IsEmpty()** (p. ??).

## Parameters

<i>hGeom</i>	The Geometry to test.
--------------	-----------------------

## Returns

TRUE if the geometry has no points, otherwise FALSE.

**13.12.2.154 int OGR\_G\_IsRing ( OGRGeometryH *hGeom* )**

Test if the geometry is a ring.

This function is the same as the C++ method **OGRGeometry::IsRing()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always return FALSE.

## Parameters

<i>hGeom</i>	The Geometry to test.
--------------	-----------------------

## Returns

TRUE if the geometry has no points, otherwise FALSE.

**13.12.2.155 int OGR\_G\_IsSimple ( OGRGeometryH *hGeom* )**

Returns TRUE if the geometry is simple.

Returns TRUE if the geometry has no anomalous geometric points, such as self intersection or self tangency. The description of each instantiable geometric class will include the specific conditions that cause an instance of that class to be classified as not simple.

This function is the same as the c++ method **OGRGeometry::IsSimple()** (p. ??) method.

If OGR is built without the GEOS library, this function will always return FALSE.

## Parameters

<i>hGeom</i>	The Geometry to test.
--------------	-----------------------

## Returns

TRUE if object is simple, otherwise FALSE.

**13.12.2.156 int OGR\_G\_IsValid ( OGRGeometryH *hGeom* )**

Test if the geometry is valid.

This function is the same as the C++ method **OGRGeometry::IsValid()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always return FALSE.

## Parameters

---

<i>hGeom</i>	The Geometry to test.
--------------	-----------------------

**Returns**

TRUE if the geometry has no points, otherwise FALSE.

**13.12.2.157 double OGR\_G\_Length ( OGRGeometryH *hGeom* )**

Compute length of a geometry.

Computes the area for **OGRCurve** (p. ??) or MultiCurve objects. Undefined for all other geometry types (returns zero).

This function utilizes the C++ get\_Length() method.

**Parameters**

<i>hGeom</i>	the geometry to operate on.
--------------	-----------------------------

**Returns**

the lenght or 0.0 for unsupported geometry types.

**Since**

OGR 1.8.0

References CPL\_Error(), wkbGeometryCollection, wkbLinearRing, wkbLineString, and wkbMultiLineString.

**13.12.2.158 int OGR\_G\_Overlaps ( OGRGeometryH *hThis*, OGRGeometryH *hOther* )**

Test for overlap.

Tests if this geometry and the other geometry overlap, that is their intersection has a non-zero area.

This function is the same as the C++ method **OGRGeometry::Overlaps()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPL\_NotSupported error.

**Parameters**

<i>hThis</i>	the geometry to compare.
<i>hOther</i>	the other geometry to compare.

**Returns**

TRUE if they are overlapping, otherwise FALSE.

**13.12.2.159 OGRGeometryH OGR\_G\_PointOnSurface ( OGRGeometryH *hGeom* )**

Returns a point guaranteed to lie on the surface.

This method relates to the SFCOM ISurface::get\_PointOnSurface() method however the current implementation based on GEOS can operate on other geometry types than the types that are supported by SQL/MM-Part 3 : surfaces (polygons) and multisurfaces (multipolygons).

This method is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this method will always fail, issuing a CPL\_NotSupported error.

## Parameters

<i>hGeom</i>	the geometry to operate on.
--------------	-----------------------------

## Returns

a point guaranteed to lie on the surface or NULL if an error occurred.

## Since

OGR 1.10

References `OGRGeometry::assignSpatialReference()`, `CPL::Error()`, `OGRGeometry::getGeometryType()`, `OGRGeometry::getSpatialReference()`, and `wkbPoint`.

Referenced by `OGRPolygon::PointOnSurface()`.

### 13.12.2.160 `OGRGeometryH OGR_G_Polygonize ( OGRGeometryH hTarget )`

Polygonizes a set of sparse edges.

A new geometry object is created and returned containing a collection of reassembled Polygons: NULL will be returned if the input collection doesn't correspond to a MultiLineString, or when reassembling Edges into Polygons is impossible due to topological inconsistencies.

This function is the same as the C++ method **`OGRGeometry::Polygonize()`** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a `CPL::Error` error.

## Parameters

<i>hTarget</i>	The Geometry to be polygonized.
----------------	---------------------------------

## Returns

a handle to a newly allocated geometry now owned by the caller, or NULL on failure.

## Since

OGR 1.9.0

### 13.12.2.161 `OGR::OGR_G_RemoveGeometry ( OGRGeometryH hGeom, int iGeom, int bDelete )`

Remove a geometry from an existing geometry container.

Removing a geometry will cause the geometry count to drop by one, and all "higher" geometries will shuffle down one in index.

There is no SFCOM analog to this method.

This function is the same as the C++ method **`OGRGeometryCollection::removeGeometry()`** (p. ??).

## Parameters

<i>hGeom</i>	the existing geometry to delete from.
--------------	---------------------------------------



<i>iGeom</i>	the index of the geometry to delete. A value of -1 is a special flag meaning that all geometries should be removed.
<i>bDelete</i>	if TRUE the geometry will be destroyed, otherwise it will not. The default is TRUE as the existing geometry is considered to own the geometries in it.

**Returns**

OGRERR\_NONE if successful, or OGRERR\_FAILURE if the index is out of range.

References CPLError(), wkbGeometryCollection, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, and wkbPolygon.

### 13.12.2.162 void OGR\_G\_Segmentize ( OGRGeometryH *hGeom*, double *dfMaxLength* )

Modify the geometry such it has no segment longer then the given distance.

Interpolated points will have Z and M values (if needed) set to 0. Distance computation is performed in 2d only

This function is the same as the CPP method **OGRGeometry::segmentize()** (p. ??).

**Parameters**

<i>hGeom</i>	handle on the geometry to segmentize
<i>dfMaxLength</i>	the maximum distance between 2 points after segmentization

References CPLError().

### 13.12.2.163 void OGR\_G\_SetCoordinateDimension ( OGRGeometryH *hGeom*, int *nNewDimension* )

Set the coordinate dimension.

This method sets the explicit coordinate dimension. Setting the coordinate dimension of a geometry to 2 should zero out any existing Z values. Setting the dimension of a geometry collection will not necessarily affect the children geometries.

**Parameters**

<i>hGeom</i>	handle on the geometry to set the dimension of the coordinates.
<i>nNewDimension</i>	New coordinate dimension value, either 2 or 3.

### 13.12.2.164 void OGR\_G\_SetPoint ( OGRGeometryH *hGeom*, int *i*, double *dfX*, double *dfY*, double *dfZ* )

Set the location of a vertex in a point or linestring geometry.

If *i*Point is larger than the number of existing points in the linestring, the point count will be increased to accomodate the request.

**Parameters**

<i>hGeom</i>	handle to the geometry to add a vertex to.
<i>i</i>	the index of the vertex to assign (zero based) or zero for a point.
<i>dfX</i>	input X coordinate to assign.
<i>dfY</i>	input Y coordinate to assign.
<i>dfZ</i>	input Z coordinate to assign (defaults to zero).

References CPLError(), wkbLineString, and wkbPoint.

### 13.12.2.165 void OGR\_G\_SetPoint\_2D ( OGRGeometryH *hGeom*, int *i*, double *dfX*, double *dfY* )

Set the location of a vertex in a point or linestring geometry.

If `iPoint` is larger than the number of existing points in the linestring, the point count will be increased to accomodate the request.

#### Parameters

<i>hGeom</i>	handle to the geometry to add a vertex to.
<i>i</i>	the index of the vertex to assign (zero based) or zero for a point.
<i>dfX</i>	input X coordinate to assign.
<i>dfY</i>	input Y coordinate to assign.

References `CPLEError()`, `wkbLineString`, and `wkbPoint`.

#### 13.12.2.166 OGRGeometryH OGR\_G\_Simplify ( OGRGeometryH *hThis*, double *dTolerance* )

Compute a simplified geometry.

This function is the same as the C++ method **OGRGeometry::Simplify()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a `CPLE_NotSupported` error.

#### Parameters

<i>hThis</i>	the geometry.
<i>dTolerance</i>	the distance tolerance for the simplification.

#### Returns

the simplified geometry or NULL if an error occurs.

#### Since

OGR 1.8.0

#### 13.12.2.167 OGRGeometryH OGR\_G\_SimplifyPreserveTopology ( OGRGeometryH *hThis*, double *dTolerance* )

Simplify the geometry while preserving topology.

This function is the same as the C++ method **OGRGeometry::SimplifyPreserveTopology()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a `CPLE_NotSupported` error.

#### Parameters

<i>hThis</i>	the geometry.
<i>dTolerance</i>	the distance tolerance for the simplification.

#### Returns

the simplified geometry or NULL if an error occurs.

#### Since

OGR 1.9.0

#### 13.12.2.168 OGRGeometryH OGR\_G\_SymDifference ( OGRGeometryH *hThis*, OGRGeometryH *hOther* )

Compute symmetric difference.

Generates a new geometry which is the symmetric difference of this geometry and the other geometry.

This function is the same as the C++ method **OGRGeometry::SymmetricDifference()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

#### Parameters

<i>hThis</i>	the geometry.
<i>hOther</i>	the other geometry.

#### Returns

a new geometry representing the symmetric difference or NULL if the difference is empty or an error occurs.

#### Since

OGR 1.8.0

13.12.2.169 OGRGeometryH OGR\_G\_SymmetricDifference ( OGRGeometryH *hThis*, OGRGeometryH *hOther* )

Compute symmetric difference (deprecated)

#### Deprecated

#### See Also

**OGR\_G\_SymmetricDifference()** (p. ??)

13.12.2.170 int OGR\_G\_Touches ( OGRGeometryH *hThis*, OGRGeometryH *hOther* )

Test for touching.

Tests if this geometry and the other geometry are touching.

This function is the same as the C++ method **OGRGeometry::Touches()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

#### Parameters

<i>hThis</i>	the geometry to compare.
<i>hOther</i>	the other geometry to compare.

#### Returns

TRUE if they are touching, otherwise FALSE.

13.12.2.171 OGRErr OGR\_G\_Transform ( OGRGeometryH *hGeom*, OGRCoordinateTransformationH *hTransform* )

Apply arbitrary coordinate transformation to geometry.

This function will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

Note that this function does not require that the geometry already have a spatial reference system. It will be assumed that they can be treated as having the source spatial reference system of the **OGRCoordinateTransformation** (p. ??) object, and the actual SRS of the geometry will be ignored. On successful completion the output **OGRSpatialReference** (p. ??) of the **OGRCoordinateTransformation** (p. ??) will be assigned to the geometry.

This function is the same as the CPP method **OGRGeometry::transform** (p. ??).

## Parameters

<i>hGeom</i>	handle on the geometry to apply the transform to.
<i>hTransform</i>	handle on the transformation to apply.

## Returns

OGRERR\_NONE on success or an error code.

13.12.2.172 OGRErr OGR\_G\_TransformTo ( OGRGeometryH *hGeom*, OGRSpatialReferenceH *hSRS* )

Transform geometry to new spatial reference system.

This function will transform the coordinates of a geometry from their current spatial reference system to a new target spatial reference system. Normally this means reprojecting the vectors, but it could include datum shifts, and changes of units.

This function will only work if the geometry already has an assigned spatial reference system, and if it is transformable to the target coordinate system.

Because this function requires internal creation and initialization of an **OGRCoordinateTransformation** (p. ??) object it is significantly more expensive to use this function to transform many geometries than it is to create the **OGRCoordinateTransformation** (p. ??) in advance, and call transform() with that transformation. This function exists primarily for convenience when only transforming a single geometry.

This function is the same as the CPP method **OGRGeometry::transformTo** (p. ??).

## Parameters

<i>hGeom</i>	handle on the geometry to apply the transform to.
<i>hSRS</i>	handle on the spatial reference system to apply.

## Returns

OGRERR\_NONE on success, or an error code.

13.12.2.173 OGRGeometryH OGR\_G\_Union ( OGRGeometryH *hThis*, OGRGeometryH *hOther* )

Compute union.

Generates a new geometry which is the region of union of the two geometries operated on.

This function is the same as the C++ method **OGRGeometry::Union()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>hThis</i>	the geometry.
<i>hOther</i>	the other geometry.

## Returns

a new geometry representing the union or NULL if an error occurs.

13.12.2.174 OGRGeometryH OGR\_G\_UnionCascaded ( OGRGeometryH *hThis* )

Compute union using cascading.

This function is the same as the C++ method **OGRGeometry::UnionCascaded()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>hThis</i>	the geometry.
--------------	---------------

## Returns

a new geometry representing the union or NULL if an error occurs.

**13.12.2.175** `int OGR_G_Within ( OGRGeometryH hThis, OGRGeometryH hOther )`

Test for containment.

Tests if this geometry is within the other geometry.

This function is the same as the C++ method **OGRGeometry::Within()** (p. ??).

This function is built on the GEOS library, check it for the definition of the geometry operation. If OGR is built without the GEOS library, this function will always fail, issuing a CPLE\_NotSupported error.

## Parameters

<i>hThis</i>	the geometry to compare.
<i>hOther</i>	the other geometry to compare.

## Returns

TRUE if *hThis* is within *hOther*, otherwise FALSE.

**13.12.2.176** `int OGR_G_WkbSize ( OGRGeometryH hGeom )`

Returns size of related binary representation.

This function returns the exact number of bytes required to hold the well known binary representation of this geometry object. Its computation may be slightly expensive for complex geometries.

This function relates to the SFCOM IWks::WkbSize() method.

This function is the same as the CPP method **OGRGeometry::WkbSize()** (p. ??).

## Parameters

<i>hGeom</i>	handle on the geometry to get the binary size from.
--------------	---

## Returns

size of binary representation in bytes.

**13.12.2.177** `const char* OGR_GetFieldName ( OGRFieldType eType )`

Fetch human readable name for a field type.

This function is the same as the CPP method **OGRFieldDefn::GetFieldName()** (p. ??).

## Parameters

<i>eType</i>	the field type to get name for.
--------------	---------------------------------

**Returns**

the name.

References `OGRFieldDefn::GetFieldTypeName()`.

### 13.12.2.178 `OGRERR OGR_L_AlterFieldDefn ( OGRLayerH hLayer, int iField, OGRFieldDefnH hNewFieldDefn, 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** (p. ??) for the layer will be updated to reflect the altered field. Applications should never modify the **OGRFeatureDefn** (p. ??) used by a layer directly.

This function 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 function. You can query a layer to check if it supports it with the `OLCAAlterFieldDefn` 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++ method **OGRLayer::AlterFieldDefn()** (p. ??).

**Parameters**

<i>hLayer</i>	handle to the layer.
<i>iField</i>	index of the field whose definition must be altered.
<i>hNewFieldDefn</i>	new field definition
<i>nFlags</i>	combination of <code>ALTER_NAME_FLAG</code> , <code>ALTER_TYPE_FLAG</code> and <code>ALTER_WIDTH_PRECISION_FLAG</code> 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

### 13.12.2.179 `OGRERR OGR_L_Clip ( OGRLayerH pLayerInput, OGRLayerH pLayerMethod, OGRLayerH pLayerResult, char ** papszOptions, GDALProgressFunc pfnProgress, void * pProgressArg )`

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.
- 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 function is the same as the C++ method **OGRLayer::Clip()** (p. ??).

#### Parameters

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

#### Since

OGR 1.10

#### 13.12.2.180 OGRErr OGR\_L\_CommitTransaction ( OGRLayerH hLayer )

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++ method **OGRLayer::CommitTransaction()**.

#### Parameters

<i>hLayer</i>	handle to the layer
---------------	---------------------

#### Returns

OGRERR\_NONE on success.

#### 13.12.2.181 OGRErr OGR\_L\_CreateFeature ( OGRLayerH hLayer, OGRFeatureH hFeat )

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 function is the same as the C++ method **OGRLayer::CreateFeature()** (p. ??).

## Parameters

<i>hLayer</i>	handle to the layer to write the feature to.
<i>hFeat</i>	the handle of the feature to write to disk.

## Returns

OGRERR\_NONE on success.

### 13.12.2.182 OGRErr OGR\_L\_CreateField ( OGRLayerH *hLayer*, OGRFieldDefnH *hField*, int *bApproxOK* )

Create a new field on a layer.

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

This function 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 function. 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++ method **OGRLayer::CreateField()** (p. ??).

## Parameters

<i>hLayer</i>	handle to the layer to write the field definition.
<i>hField</i>	handle of the field definition to write to disk.
<i>bApproxOK</i>	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.

### 13.12.2.183 OGRErr OGR\_L\_DeleteFeature ( OGRLayerH *hLayer*, long *nFID* )

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 **OGR\_L\_TestCapability()** (p. ??) function may be called with **OLCDeleteFeature** to check if the driver supports feature deletion.

This method is the same as the C++ method **OGRLayer::DeleteFeature()** (p. ??).

## Parameters

<i>hLayer</i>	handle to the layer
<i>nFID</i>	the feature id to be deleted from the layer

## Returns

OGRERR\_NONE on success.

### 13.12.2.184 OGRErr OGR\_L\_DeleteField ( OGRLayerH *hLayer*, int *iField* )

Create a new field on a layer.

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

This function 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 function. 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++ method **OGRLayer::DeleteField()** (p. ??).

#### Parameters

<i>hLayer</i>	handle to the layer.
<i>iField</i>	index of the field to delete.

#### Returns

`OGRERR_NONE` on success.

#### Since

OGR 1.9.0

**13.12.2.185** `OGRERR OGR_L_Erase ( OGRLayerH pLayerInput, OGRLayerH pLayerMethod, OGRLayerH pLayerResult, char ** ppszOptions, GDALProgressFunc pfnProgress, void * pProgressArg )`

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.
- `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 function is the same as the C++ method **OGRLayer::Erase()** (p. ??).

#### Parameters

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

#### Since

OGR 1.10

#### 13.12.2.186 OGRErr OGR\_L\_GetExtent ( OGRLayerH hLayer, OGREnvelope \* psExtent, int bForce )

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 **OGR\_L\_GetExtent()** (p. ??) 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 function is the same as the C++ method **OGRLayer::GetExtent()** (p. ??).

#### Parameters

<i>hLayer</i>	handle to the layer from which to get extent.
<i>psExtent</i>	the structure in which the extent value will be returned.
<i>bForce</i>	Flag indicating whether the extent should be computed even if it is expensive.

#### Returns

OGRERR\_NONE on success, OGRERR\_FAILURE if extent not known.

#### 13.12.2.187 OGRFeatureH OGR\_L\_GetFeature ( OGRLayerH hLayer, long nFeatureId )

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.

If this function returns a non-NULL feature, it is guaranteed that its feature id (**OGR\_F\_GetFID()** (p. ??)) will be the same as nFID.

Use **OGR\_L\_TestCapability(OLCRandomRead)** to establish if this layer supports efficient random access reading via **OGR\_L\_GetFeature()** (p. ??); 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 are generally considered interrupted by a **OGR\_L\_GetFeature()** (p. ??) call.

The returned feature should be free with **OGR\_F\_Destroy()** (p. ??).

This function is the same as the C++ method **OGRLayer::GetFeature()** (p. ??).

## Parameters

<i>hLayer</i>	handle to the layer that owned the feature.
<i>nFeatureId</i>	the feature id of the feature to read.

## Returns

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

13.12.2.188 int OGR\_L\_GetFeatureCount ( OGRLayerH *hLayer*, int *bForce* )

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 function is the same as the CPP **OGRLayer::GetFeatureCount()** (p. ??).

## Parameters

<i>hLayer</i>	handle to the layer that owned the features.
<i>bForce</i>	Flag indicating whether the count should be computed even if it is expensive.

## Returns

feature count, -1 if count not known.

13.12.2.189 const char \* OGR\_L\_GetFIDColumn ( OGRLayerH *hLayer* )

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++ method **OGRLayer::GetFIDColumn()** (p. ??)

## Parameters

<i>hLayer</i>	handle to the layer
---------------	---------------------

## Returns

fid column name.

13.12.2.190 const char \* OGR\_L\_GetGeometryColumn ( OGRLayerH *hLayer* )

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++ method **OGRLayer::GetGeometryColumn()** (p. ??)

## Parameters

<i>hLayer</i>	handle to the layer
---------------	---------------------

**Returns**

geometry column name.

**13.12.2.191 OGRwkbGeometryType OGR\_L\_GetGeomType ( OGRLayerH hLayer )**

Return the layer geometry type.

This returns the same result as `OGR_FD_GetGeomType(OGR_L_GetLayerDefn(hLayer))`, but for a few drivers, calling **OGR\_L\_GetGeomType()** (p. ??) directly can avoid lengthy layer definition initialization.

This function is the same as the C++ method **OGRLayer::GetGeomType()** (p. ??).

**Parameters**

<i>hLayer</i>	handle to the layer.
---------------	----------------------

**Returns**

the geometry type

**Since**

OGR 1.8.0

References wkbUnknown.

**13.12.2.192 OGRFeatureDefnH OGR\_L\_GetLayerDefn ( OGRLayerH hLayer )**

Fetch the schema information for this layer.

The returned handle to the **OGRFeatureDefn** (p. ??) is owned by the **OGRLayer** (p. ??), and should not be modified or freed by the application. It encapsulates the attribute schema of the features of the layer.

This function is the same as the C++ method **OGRLayer::GetLayerDefn()** (p. ??).

**Parameters**

<i>hLayer</i>	handle to the layer to get the schema information.
---------------	--

**Returns**

an handle to the feature definition.

**13.12.2.193 const char \* OGR\_L\_GetName ( OGRLayerH hLayer )**

Return the layer name.

This returns the same content as `OGR_FD_GetName(OGR_L_GetLayerDefn(hLayer))`, but for a few drivers, calling **OGR\_L\_GetName()** (p. ??) directly can avoid lengthy layer definition initialization.

This function is the same as the C++ method **OGRLayer::GetName()** (p. ??).

## Parameters

<i>hLayer</i>	handle to the layer.
---------------	----------------------

## Returns

the layer name (must not be freed)

## Since

OGR 1.8.0

13.12.2.194 OGRFeatureH OGR\_L\_GetNextFeature ( OGRLayerH *hLayer* )

Fetch the next available feature from this layer.

The returned feature becomes the responsibility of the caller to delete with **OGR\_F\_Destroy()** (p. ??). It is critical that all features associated with an **OGRLayer** (p. ??) (more specifically an **OGRFeatureDefn** (p. ??)) be deleted before that layer/datasource is deleted.

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

This function implements sequential access to the features of a layer. The **OGR\_L\_ResetReading()** (p. ??) function can be used to start at the beginning again.

This function is the same as the C++ method **OGRLayer::GetNextFeature()** (p. ??).

## Parameters

<i>hLayer</i>	handle to the layer from which feature are read.
---------------	--

## Returns

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

13.12.2.195 OGRGeometryH OGR\_L\_GetSpatialFilter ( OGRLayerH *hLayer* )

This function 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 function is the same as the C++ method **OGRLayer::GetSpatialFilter()** (p. ??).

## Parameters

<i>hLayer</i>	handle to the layer to get the spatial filter from.
---------------	---

## Returns

an handle to the spatial filter geometry.

13.12.2.196 OGRSpatialReferenceH OGR\_L\_GetSpatialRef ( OGRLayerH *hLayer* )

Fetch the spatial reference system for this layer.

The returned object is owned by the **OGRLayer** (p. ??) and should not be modified or freed by the application.

This function is the same as the C++ method **OGRLayer::GetSpatialRef()** (p. ??).

## Parameters

<i>hLayer</i>	handle to the layer to get the spatial reference from.
---------------	--

## Returns

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

13.12.2.197 **OGRERR** **OGR\_L\_Identity** ( **OGR**LayerH *pLayerInput*, **OGR**LayerH *pLayerMethod*, **OGR**LayerH *pLayerResult*, char \*\* *papszOptions*, **GDAL**ProgressFunc *pfnProgress*, void \* *pProgressArg* )

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.
- **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 function is the same as the C++ method **OGR**Layer::Identity() (p. ??).

## Parameters

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

## Since

OGR 1.10



13.12.2.198 OGRErr OGR\_L\_Intersection ( OGRLayerH *pLayerInput*, OGRLayerH *pLayerMethod*, OGRLayerH *pLayerResult*, char \*\* *papszOptions*, GDALProgressFunc *pfnProgress*, void \* *pProgressArg* )

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 it to YES to go on, even when a feature could not be inserted.
- 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 function is the same as the C++ method **OGRLayer::Intersection()** (p. ??).

#### Parameters

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

#### Since

OGR 1.10

13.12.2.199 OGRErr OGR\_L\_ReorderField ( OGRLayerH *hLayer*, int *iOldFieldPos*, int *iNewFieldPos* )

Reorder an existing field on a layer.

This function is a conveniency wrapper of **OGR\_L\_ReorderFields()** (p. ??) dedicated to move a single field.

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

This function 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 function. 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++ method **`OGRLayer::ReorderField()`** (p. ??).

#### Parameters

<i>hLayer</i>	handle to the layer.
<i>iOldFieldPos</i>	previous position of the field to move. Must be in the range <code>[0,GetFieldCount()-1]</code> .
<i>iNewFieldPos</i>	new position of the field to move. Must be in the range <code>[0,GetFieldCount()-1]</code> .

#### Returns

`OGRERR_NONE` on success.

#### Since

OGR 1.9.0

### 13.12.2.200 **`OGRERR OGR_L_ReorderFields ( OGRLayerH hLayer, int * panMap )`**

Reorder all the fields of a layer.

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

This function 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 function. 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++ method **`OGRLayer::ReorderFields()`** (p. ??).

#### Parameters

<i>hLayer</i>	handle to the layer.
<i>panMap</i>	an array of <code>GetLayerDefn()-&gt;GetFieldCount()</code> elements which is a permutation of <code>[0, GetLayerDefn()-&gt;GetFieldCount()-1]</code> .

#### Returns

`OGRERR_NONE` on success.

#### Since

OGR 1.9.0

**13.12.2.201 void OGR\_L\_ResetReading ( OGRLayerH *hLayer* )**

Reset feature reading to start on the first feature.

This affects GetNextFeature().

This function is the same as the C++ method **OGRLayer::ResetReading()** (p. ??).

**Parameters**

<i>hLayer</i>	handle to the layer on which features are read.
---------------	---

**13.12.2.202 OGRErr OGR\_L\_RollbackTransaction ( OGRLayerH *hLayer* )**

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++ method OGRLayer::RollbackTransaction().

**Parameters**

<i>hLayer</i>	handle to the layer
---------------	---------------------

**Returns**

OGRErr\_NONE on success.

**13.12.2.203 OGRErr OGR\_L\_SetAttributeFilter ( OGRLayerH *hLayer*, const char \* *pszQuery* )**

Set a new attribute query.

This function sets the attribute query string to be used when fetching features via the **OGR\_L\_GetNextFeature()** (p. ??) function. 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 a restricted form of SQL WHERE clause as defined "eq\_format=restricted\_where" about half way through this document:

<http://ogdi.sourceforge.net/prop/6.2.CapabilitiesMetadata.html>

Note that installing a query string will generally result in resetting the current reading position (ala **OGR\_L\_ResetReading()** (p. ??)).

This function is the same as the C++ method **OGRLayer::SetAttributeFilter()** (p. ??).

**Parameters**

<i>hLayer</i>	handle to the layer on which attribute query will be executed.
<i>pszQuery</i>	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.

**13.12.2.204 OGRErr OGR\_L\_SetFeature ( OGRLayerH *hLayer*, OGRFeatureH *hFeat* )**

Rewrite an existing feature.

This function will write a feature to the layer, based on the feature id within the **OGRFeature** (p. ??).

Use `OGR_L_TestCapability(OLCRandomWrite)` to establish if this layer supports random access writing via **`OGR_L_SetFeature()`** (p. ??).

This function is the same as the C++ method **`OGRLayer::SetFeature()`** (p. ??).

#### Parameters

<i>hLayer</i>	handle to the layer to write the feature.
<i>hFeat</i>	the feature to write.

#### Returns

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

#### 13.12.2.205 `OGRERR OGR_L_SetIgnoredFields ( OGRLayerH , const char ** papszFields )`

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++ method **`OGRLayer::SetIgnoredFields()`** (p. ??)

#### Parameters

<i>papszFields</i>	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)

#### 13.12.2.206 `OGRERR OGR_L_SetNextByIndex ( OGRLayerH hLayer, long nIndex )`

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++ method **`OGRLayer::SetNextByIndex()`** (p. ??)

#### Parameters

<i>hLayer</i>	handle to the layer
<i>nIndex</i>	the index indicating how many steps into the result set to seek.

#### Returns

`OGRERR_NONE` on success or an error code.

**13.12.2.207 void OGR\_L\_SetSpatialFilter ( OGRLayerH *hLayer*, OGRGeometryH *hGeom* )**

Set a new spatial filter.

This function set the geometry to be used as a spatial filter when fetching features via the **OGR\_L\_GetNextFeature()** (p. ??) function. 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 **OGR\_G\_GetEnvelope()** (p. ??)) 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 function 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 **OGR\_L\_GetSpatialRef()** (p. ??)). In the future this may be generalized.

This function is the same as the C++ method **OGRLayer::SetSpatialFilter** (p. ??).

**Parameters**

<i>hLayer</i>	handle to the layer on which to set the spatial filter.
<i>hGeom</i>	handle to 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.

**13.12.2.208 void OGR\_L\_SetSpatialFilterRect ( OGRLayerH *hLayer*, double *dfMinX*, double *dfMinY*, double *dfMaxX*, double *dfMaxY* )**

Set a new rectangular spatial filter.

This method set rectangle to be used as a spatial filter when fetching features via the **OGR\_L\_GetNextFeature()** (p. ??) 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()** (p. ??)). Internally this method is normally implemented as creating a 5 vertex closed rectangular polygon and passing it to **OGRLayer::SetSpatialFilter()** (p. ??). 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++ method **OGRLayer::SetSpatialFilterRect()** (p. ??).

**Parameters**

<i>hLayer</i>	handle to the layer on which to set the spatial filter.
<i>dfMinX</i>	the minimum X coordinate for the rectangular region.
<i>dfMinY</i>	the minimum Y coordinate for the rectangular region.
<i>dfMaxX</i>	the maximum X coordinate for the rectangular region.
<i>dfMaxY</i>	the maximum Y coordinate for the rectangular region.

**13.12.2.209 OGRErr OGR\_L\_StartTransaction ( OGRLayerH *hLayer* )**

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.

This function is the same as the C++ method **OGRLayer::StartTransaction()**.

**Parameters**

<i>hLayer</i>	handle to the layer
---------------	---------------------

#### Returns

OGRERR\_NONE on success.

**13.12.2.210** `OGRERR OGR_L_SymDifference ( OGRLayerH pLayerInput, OGRLayerH pLayerMethod, OGRLayerH 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.
- 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 function is the same as the C++ method **OGRLayer::SymDifference()** (p. ??).

#### Parameters

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

#### Since

OGR 1.10

13.12.2.211 OGRErr OGR\_L\_SyncToDisk ( OGRLayerH *hLayer* )

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 **OGR\_DS\_Destroy()** (p. ??) that will ensure all data is correctly flushed.

This method is the same as the C++ method **OGRLayer::SyncToDisk()** (p. ??)

## Parameters

<i>hLayer</i>	handle to the layer
---------------	---------------------

## Returns

OGRERR\_NONE if no error occurs (even if nothing is done) or an error code.

13.12.2.212 int OGR\_L\_TestCapability ( OGRLayerH *hLayer*, const char \* *pszCap* )

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** (p. ??) 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** (p. ??) 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** (p. ??) 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 **OGR\_L\_GetFeatureCount()** (p. ??)) 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 **OGR\_L\_GetExtent()** (p. ??)) 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.
- **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.
- **OLCAAlterFieldDefn** / "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.

This function is the same as the C++ method **OGRLayer::TestCapability()** (p. ??).

#### Parameters

<i>hLayer</i>	handle to the layer to get the capability from.
<i>pszCap</i>	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.

**13.12.2.213 OGRErr OGR\_L\_Union ( OGRLayerH pLayerInput, OGRLayerH pLayerMethod, OGRLayerH pLayerResult, char \*\* ppszOptions, GDALProgressFunc pfnProgress, void \* pProgressArg )**

Union 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. 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.
- **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 function is the same as the C++ method **OGRLayer::Union()** (p. ??).



## Parameters

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

## Since

OGR 1.10

**13.12.2.214** `OGRErr OGR_L_Update ( OGRLayerH pLayerInput, OGRLayerH pLayerMethod, OGRLayerH pLayerResult, char ** papszOptions, GDALProgressFunc pfnProgress, void * pProgressArg )`

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 areas 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 that 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.
- 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 function is the same as the C++ method **OGRLayer::Update()** (p. ??).

## Parameters

<i>pLayerInput</i>	the input layer. Should not be NULL.
--------------------	--------------------------------------

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

#### Since

OGR 1.10

13.12.2.215 `int OGR_SM_AddPart ( OGRStyleMgrH hSM, OGRStyleToolH hST )`

Add a part (style tool) to the current style.

This function is the same as the C++ method **OGRStyleMgr::AddPart()** (p. ??).

#### Parameters

<i>hSM</i>	handle to the style manager.
<i>hST</i>	the style tool defining the part to add.

#### Returns

TRUE on success, FALSE on errors.

13.12.2.216 `int OGR_SM_AddStyle ( OGRStyleMgrH hSM, const char * pszStyleName, const char * pszStyleString )`

Add a style to the current style table.

This function is the same as the C++ method **OGRStyleMgr::AddStyle()** (p. ??).

#### Parameters

<i>hSM</i>	handle to the style manager.
<i>pszStyleName</i>	the name of the style to add.
<i>pszStyleString</i>	the style string to use, or NULL to use the style stored in the manager.

#### Returns

TRUE on success, FALSE on errors.

13.12.2.217 `OGRStyleMgrH OGR_SM_Create ( OGRStyleTableH hStyleTable )`

**OGRStyleMgr** (p. ??) factory.

This function is the same as the C++ method **OGRStyleMgr::OGRStyleMgr()** (p. ??).

## Parameters

<i>hStyleTable</i>	pointer to <b>OGRStyleTable</b> (p. ??) or NULL if not working with a style table.
--------------------	--

## Returns

an handle to the new style manager object.

13.12.2.218 void OGR\_SM\_Destroy ( OGRStyleMgrH *hSM* )

Destroy Style Manager.

This function is the same as the C++ method **OGRStyleMgr::~OGRStyleMgr()** (p. ??).

## Parameters

<i>hSM</i>	handle to the style manager to destroy.
------------	---

13.12.2.219 OGRStyleToolH OGR\_SM\_GetPart ( OGRStyleMgrH *hSM*, int *nPartId*, const char \* *pszStyleString* )

Fetch a part (style tool) from the current style.

This function is the same as the C++ method **OGRStyleMgr::GetPart()** (p. ??).

This function instanciates a new object that should be freed with **OGR\_ST\_Destroy()** (p. ??).

## Parameters

<i>hSM</i>	handle to the style manager.
<i>nPartId</i>	the part number (0-based index).
<i>pszStyleString</i>	(optional) the style string on which to operate. If NULL then the current style string stored in the style manager is used.

## Returns

OGRStyleToolH of the requested part (style tools) or NULL on error.

13.12.2.220 int OGR\_SM\_GetPartCount ( OGRStyleMgrH *hSM*, const char \* *pszStyleString* )

Get the number of parts in a style.

This function is the same as the C++ method **OGRStyleMgr::GetPartCount()** (p. ??).

## Parameters

<i>hSM</i>	handle to the style manager.
<i>pszStyleString</i>	(optional) the style string on which to operate. If NULL then the current style string stored in the style manager is used.

## Returns

the number of parts (style tools) in the style.

13.12.2.221 const char\* OGR\_SM\_InitFromFeature ( OGRStyleMgrH *hSM*, OGRFeatureH *hFeat* )

Initialize style manager from the style string of a feature.

This function is the same as the C++ method **OGRStyleMgr::InitFromFeature()** (p. ??).

## Parameters

<i>hSM</i>	handle to the style manager.
<i>hFeat</i>	handle to the new feature from which to read the style.

## Returns

a reference to the style string read from the feature, or NULL in case of error.

13.12.2.222 `int OGR_SM_InitStyleString ( OGRStyleMgrH hSM, const char * pszStyleString )`

Initialize style manager from the style string.

This function is the same as the C++ method **OGRStyleMgr::InitStyleString()** (p. ??).

## Parameters

<i>hSM</i>	handle to the style manager.
<i>pszStyleString</i>	the style string to use (can be NULL).

## Returns

TRUE on success, FALSE on errors.

13.12.2.223 `OGRStyleToolH OGR_ST_Create ( OGRSTClassId eClassId )`

**OGRStyleTool** (p. ??) factory.

This function is a constructor for **OGRStyleTool** (p. ??) derived classes.

## Parameters

<i>eClassId</i>	subclass of style tool to create. One of OGRSTCPen (1), OGRSTCBrush (2), OGRSTC-Symbol (3) or OGRSTCLabel (4).
-----------------	--

## Returns

an handle to the new style tool object or NULL if the creation failed.

13.12.2.224 `void OGR_ST_Destroy ( OGRStyleToolH hST )`

Destroy Style Tool.

## Parameters

<i>hST</i>	handle to the style tool to destroy.
------------	--------------------------------------

13.12.2.225 `double OGR_ST_GetParamDbl ( OGRStyleToolH hST, int eParam, int * bValueIsNull )`

Get Style Tool parameter value as a double.

Maps to the **OGRStyleTool** (p. ??) subclasses' GetParamDbl() methods.

## Parameters

<i>hST</i>	handle to the style tool.
<i>eParam</i>	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
<i>bValueIsNull</i>	pointer to an integer that will be set to TRUE or FALSE to indicate whether the parameter value is NULL.

## Returns

the parameter value as double and sets *bValueIsNull*.

13.12.2.226 `int OGR_ST_GetParamNum ( OGRStyleToolH hST, int eParam, int * bValueIsNull )`

Get Style Tool parameter value as an integer.

Maps to the **OGRStyleTool** (p. ??) subclasses' `GetParamNum()` methods.

## Parameters

<i>hST</i>	handle to the style tool.
<i>eParam</i>	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
<i>bValueIsNull</i>	pointer to an integer that will be set to TRUE or FALSE to indicate whether the parameter value is NULL.

## Returns

the parameter value as integer and sets *bValueIsNull*.

13.12.2.227 `const char* OGR_ST_GetParamStr ( OGRStyleToolH hST, int eParam, int * bValueIsNull )`

Get Style Tool parameter value as string.

Maps to the **OGRStyleTool** (p. ??) subclasses' `GetParamStr()` methods.

## Parameters

<i>hST</i>	handle to the style tool.
<i>eParam</i>	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
<i>bValueIsNull</i>	pointer to an integer that will be set to TRUE or FALSE to indicate whether the parameter value is NULL.

## Returns

the parameter value as string and sets *bValueIsNull*.

13.12.2.228 `int OGR_ST_GetRGBFromString ( OGRStyleToolH hST, const char * pszColor, int * pnRed, int * pnGreen, int * pnBlue, int * pnAlpha )`

Return the r,g,b,a components of a color encoded in #RRGGBB[AA] format.

Maps to `OGRStyleTool::GetRGBFromString()`.

## Parameters

<i>hST</i>	handle to the style tool.
<i>pszColor</i>	the color to parse
<i>pnRed</i>	pointer to an int in which the red value will be returned
<i>pnGreen</i>	pointer to an int in which the green value will be returned
<i>pnBlue</i>	pointer to an int in which the blue value will be returned
<i>pnAlpha</i>	pointer to an int in which the (optional) alpha value will be returned

## Returns

TRUE if the color could be succesfully parsed, or FALSE in case of errors.

13.12.2.229 `const char* OGR_ST_GetStyleString ( OGRStyleToolH hST )`

Get the style string for this Style Tool.

Maps to the **OGRStyleTool** (p. ??) subclasses' GetStyleString() methods.

## Parameters

<i>hST</i>	handle to the style tool.
------------	---------------------------

## Returns

the style string for this style tool or "" if the hST is invalid.

13.12.2.230 `OGRSTCClassId OGR_ST_GetType ( OGRStyleToolH hST )`

Determine type of Style Tool.

## Parameters

<i>hST</i>	handle to the style tool.
------------	---------------------------

## Returns

the style tool type, one of OGRSTCPen (1), OGRSTCBrush (2), OGRSTCSymbol (3) or OGRSTCLabel (4).  
Returns OGRSTCNone (0) if the OGRStyleToolH is invalid.

13.12.2.231 `OGRSTUnitId OGR_ST_GetUnit ( OGRStyleToolH hST )`

Get Style Tool units.

## Parameters

<i>hST</i>	handle to the style tool.
------------	---------------------------

## Returns

the style tool units.

13.12.2.232 `void OGR_ST_SetParamDbI ( OGRStyleToolH hST, int eParam, double dfValue )`

Set Style Tool parameter value from a double.

Maps to the **OGRStyleTool** (p. ??) subclasses' SetParamDbI() methods.

## Parameters

<i>hST</i>	handle to the style tool.
<i>eParam</i>	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
<i>dfValue</i>	the new parameter value

13.12.2.233 void OGR\_ST\_SetParamNum ( OGRStyleToolH *hST*, int *eParam*, int *nValue* )

Set Style Tool parameter value from an integer.

Maps to the **OGRStyleTool** (p. ??) subclasses' SetParamNum() methods.

## Parameters

<i>hST</i>	handle to the style tool.
<i>eParam</i>	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
<i>nValue</i>	the new parameter value

13.12.2.234 void OGR\_ST\_SetParamStr ( OGRStyleToolH *hST*, int *eParam*, const char \* *pszValue* )

Set Style Tool parameter value from a string.

Maps to the **OGRStyleTool** (p. ??) subclasses' SetParamStr() methods.

## Parameters

<i>hST</i>	handle to the style tool.
<i>eParam</i>	the parameter id from the enumeration corresponding to the type of this style tool (one of the OGRSTPenParam, OGRSTBrushParam, OGRSTSymbolParam or OGRSTLabelParam enumerations)
<i>pszValue</i>	the new parameter value

13.12.2.235 void OGR\_ST\_SetUnit ( OGRStyleToolH *hST*, OGRSTUnitId *eUnit*, double *dfGroundPaperScale* )

Set Style Tool units.

This function is the same as OGRStyleTool::SetUnit()

## Parameters

<i>hST</i>	handle to the style tool.
<i>eUnit</i>	the new unit.
<i>dfGroundPaperScale</i>	ground to paper scale factor.

13.12.2.236 OGRStyleTableH OGR\_STBL\_Create ( void )

**OGRStyleTable** (p. ??) factory.

This function is the same as the C++ method OGRStyleTable::OGRStyleTable().

**Returns**

an handle to the new style table object.

13.12.2.237 `void OGR_STBL_Destroy ( OGRStyleTableH hSTBL )`

Destroy Style Table.

**Parameters**

<i>hSTBL</i>	handle to the style table to destroy.
--------------	---------------------------------------

13.12.2.238 `const char* OGR_STBL_Find ( OGRStyleTableH hStyleTable, const char * pszName )`

Get a style string by name.

This function is the same as the C++ method **OGRStyleTable::Find()** (p. ??).

**Parameters**

<i>hStyleTable</i>	handle to the style table.
<i>pszName</i>	the name of the style string to find.

**Returns**

the style string matching the name or NULL if not found or error.

13.12.2.239 `const char* OGR_STBL_GetLastStyleName ( OGRStyleTableH hStyleTable )`

Get the style name of the last style string fetched with OGR\_STBL\_GetNextStyle.

This function is the same as the C++ method **OGRStyleTable::GetStyleName()** (p. ??).

**Parameters**

<i>hStyleTable</i>	handle to the style table.
--------------------	----------------------------

**Returns**

the Name of the last style string or NULL on error.

13.12.2.240 `const char* OGR_STBL_GetNextStyle ( OGRStyleTableH hStyleTable )`

Get the next style string from the table.

This function is the same as the C++ method OGRStyleTable::GetNextStyle().

**Parameters**

<i>hStyleTable</i>	handle to the style table.
--------------------	----------------------------

**Returns**

the next style string or NULL on error.

13.12.2.241 `int OGR_STBL_LoadStyleTable ( OGRStyleTableH hStyleTable, const char * pszFilename )`

Load a style table from a file.

This function is the same as the C++ method **OGRStyleTable::LoadStyleTable()** (p. ??).



## Parameters

<i>hStyleTable</i>	handle to the style table.
<i>pszFilename</i>	the name of the file to load from.

## Returns

TRUE on success, FALSE on error

13.12.2.242 void OGR\_STBL\_ResetStyleStringReading ( OGRStyleTableH *hStyleTable* )

Reset the next style pointer to 0.

This function is the same as the C++ method OGRStyleTable::ResetStyleStringReading().

## Parameters

<i>hStyleTable</i>	handle to the style table.
--------------------	----------------------------

13.12.2.243 int OGR\_STBL\_SaveStyleTable ( OGRStyleTableH *hStyleTable*, const char \* *pszFilename* )

Save a style table to a file.

This function is the same as the C++ method **OGRStyleTable::SaveStyleTable()** (p. ??).

## Parameters

<i>hStyleTable</i>	handle to the style table.
<i>pszFilename</i>	the name of the file to save to.

## Returns

TRUE on success, FALSE on error

13.12.2.244 OGRGeometryH OGRBuildPolygonFromEdges ( OGRGeometryH *hLines*, int *bBestEffort*, int *bAutoClose*, double *dfTolerance*, OGRErr \* *peErr* )

Build a ring from a bunch of arcs.

## Parameters

<i>hLines</i>	handle to an <b>OGRGeometryCollection</b> (p. ??) (or <b>OGRMultiLineString</b> (p. ??)) containing the line string geometries to be built into rings.
<i>bBestEffort</i>	not yet implemented???
<i>bAutoClose</i>	indicates if the ring should be close when first and last points of the ring are the same.
<i>dfTolerance</i>	tolerance into which two arcs are considered close enough to be joined.
<i>peErr</i>	OGRERR_NONE on success, or OGRERR_FAILURE on failure.

## Returns

an handle to the new geometry, a polygon.

References OGRLineString::addPoint(), OGRPolygon::addRingDirectly(), CPLCalloc(), CPLDebug(), CPLError(), OGRGeometryCollection::getGeometryRef(), OGRGeometry::getGeometryType(), OGRGeometryCollection::getNumGeometries(), OGRLineString::getNumPoints(), OGRLineString::getX(), OGRLineString::getY(), OGRLineString::getZ(), wkbGeometryCollection, wkbLineString, and wkbMultiLineString.

**13.12.2.245 void OGRCleanupAll ( void )**

Cleanup all OGR related resources.

This function will destroy the **OGRSFDriverRegistrar** (p. ??) along with all registered drivers, and then cleanup long lived OSR (**OGRSpatialReference** (p. ??)) and CPL resources. This may be called in an application when OGR services are no longer needed. It is not normally required, but by freeing all dynamically allocated memory it can make memory leak testing easier.

In addition to destroying the OGRDriverRegistrar, this function also calls:

- **OSRCleanup()** (p. ??)
- **CPLFinderClean()**
- **VSICleanupFileManager()**
- **CPLFreeConfig()**
- **CPLCleanupTLS()**

References OSRCleanup().

**13.12.2.246 void OGRDeregisterDriver ( OGRSFDriverH hDriver )**

Remove the passed driver from the list of registered drivers.

This function is the same as the C++ method **OGRSFDriverRegistrar::DeregisterDriver()** (p. ??).

**Parameters**

<i>hDriver</i>	handle to the driver to deregister.
----------------	-------------------------------------

**Since**

GDAL 1.8.0

References OGRSFDriverRegistrar::DeregisterDriver(), and OGRSFDriverRegistrar::GetRegistrar().

**13.12.2.247 OGRSFDriverH OGRGetDriver ( int iDriver )**

Fetch the indicated driver.

This function is the same as the C++ method **OGRSFDriverRegistrar::GetDriver()** (p. ??).

**Parameters**

<i>iDriver</i>	the driver index, from 0 to GetDriverCount()-1.
----------------	---

**Returns**

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

References OGRSFDriverRegistrar::GetDriver().

**13.12.2.248 OGRSFDriverH OGRGetDriverByName ( const char \* pszName )**

Fetch the indicated driver.

This function is the same as the C++ method **OGRSFDriverRegistrar::GetDriverByName()** (p. ??)

## Parameters

<i>pszName</i>	the driver name
----------------	-----------------

## Returns

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

References `OGRSFDriverRegistrar::GetDriverByName()`, and `OGRSFDriverRegistrar::GetRegistrar()`.

13.12.2.249 `int OGRGetDriverCount ( void )`

Fetch the number of registered drivers.

This function is the same as the C++ method `OGRSFDriverRegistrar::GetDriverCount()` (p. ??).

## Returns

the drivers count.

References `OGRSFDriverRegistrar::GetDriverCount()`.

13.12.2.250 `OGRDataSourceH OGRGetOpenDS ( int iDS )`

Return the iDS th datasource opened.

This function is the same as the C++ method `OGRSFDriverRegistrar::GetOpenDS` (p. ??).

## Parameters

<i>iDS</i>	the index of the dataset to return (between 0 and <code>GetOpenDSCount()</code> - 1)
------------	--

References `OGRSFDriverRegistrar::GetOpenDS()`, and `OGRSFDriverRegistrar::GetRegistrar()`.

13.12.2.251 `int OGRGetOpenDSCount ( void )`

Return the number of opened datasources.

This function is the same as the C++ method `OGRSFDriverRegistrar::GetOpenDSCount()` (p. ??)

## Returns

the number of opened datasources.

References `OGRSFDriverRegistrar::GetOpenDSCount()`, and `OGRSFDriverRegistrar::GetRegistrar()`.

13.12.2.252 `OGRDataSourceH OGROpen ( const char * pszName, int bUpdate, OGRSFDriverH * pahDriverList )`

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

This function loops through all the drivers registered with the driver manager trying each until one succeeds with the given data source. This function is static. Applications don't normally need to use any other `OGRSFDriverRegistrar` (p. ??) function, not do they normally need to have a pointer to an `OGRSFDriverRegistrar` (p. ??) instance.

If this function fails, `CPLGetLastErrorMsg()` (p. ??) 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 `VSIInstallZipFileHandler()` (p. ??)), in a .tar/.tar.gz/.tgz archive (see `VSIInstallTarFileHandler()` (p. ??)) or on a HTTP / FTP server (see `VSIInstallCurlFileHandler()` (p. ??))

This function is the same as the C++ method `OGRSFDriverRegistrar::Open()` (p. ??).

## Parameters

<i>pszName</i>	the name of the file, or data source to open.
<i>bUpdate</i>	FALSE for read-only access (the default) or TRUE for read-write access.
<i>pahDriverList</i>	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 an handle to an **OGRDataSource** (p. ??). This **OGRDataSource** (p. ??) should be closed by deleting the object when it is no longer needed.

## Example:

```
OGRDataSourceH      hDS;
OGRSFDriverH       *pahDriver;

hDS = OGROpen( "polygon.shp", 0, pahDriver );
if( hDS == NULL )
{
    return;
}

... use the data source ...

OGRReleaseDataSource( hDS );
```

References OGRSFDriverRegistrar::Open().

### 13.12.2.253 void OGRRegisterDriver ( OGRSFDriverH hDriver )

Add a driver to the list of registered drivers.

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

This function is the same as the C++ method **OGRSFDriverRegistrar::RegisterDriver()** (p. ??).

## Parameters

<i>hDriver</i>	handle to the driver to add.
----------------	------------------------------

References OGRSFDriverRegistrar::GetRegistrar(), and OGRSFDriverRegistrar::RegisterDriver().

### 13.12.2.254 OGRErr OGRReleaseDataSource ( OGRDataSourceH hDS )

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++ method **OGRDataSource::Release()** (p. ??)

## Parameters

<i>hDS</i>	handle to the data source to release
------------	--------------------------------------

**Returns**

OGRERR\_NONE on success or an error code.

References OGRSFDriverRegistrar::GetRegistrar().

Referenced by OGRGeocodeDestroySession().

### 13.12.2.255 OGRErr OGRSetGenerate\_DB2\_V72\_BYTE\_ORDER ( int *bGenerate\_DB2\_V72\_BYTE\_ORDER* )

Special entry point to enable the hack for generating DB2 V7.2 style WKB.

DB2 seems to have placed (and require) an extra 0x30 or'ed with the byte order in WKB. This entry point is used to turn on or off the generation of such WKB.

## 13.13 ogr\_core.h File Reference

```
#include "cpl_port.h"
#include "gdal_version.h"
```

**Classes**

- class **OGREnvelope**
- class **OGREnvelope3D**
- union **OGRField**

**Macros**

- #define **GDAL\_CHECK\_VERSION**(pszCallingComponentName) **GDALCheckVersion**(GDAL\_VERSION\_ - MAJOR, GDAL\_VERSION\_MINOR, pszCallingComponentName)

**Typedefs**

- typedef enum  
**ogr\_style\_tool\_class\_id** OGRSTClassId
- typedef enum  
**ogr\_style\_tool\_units\_id** OGRSTUnitId
- typedef enum  
**ogr\_style\_tool\_param\_pen\_id** OGRSTPenParam
- typedef enum  
**ogr\_style\_tool\_param\_brush\_id** OGRSTBrushParam
- typedef enum  
**ogr\_style\_tool\_param\_symbol\_id** OGRSTSymbolParam
- typedef enum  
**ogr\_style\_tool\_param\_label\_id** OGRSTLabelParam

## Enumerations

- enum **OGRwkbGeometryType** {  
**wkbUnknown** = 0, **wkbPoint** = 1, **wkbLineString** = 2, **wkbPolygon** = 3,  
**wkbMultiPoint** = 4, **wkbMultiLineString** = 5, **wkbMultiPolygon** = 6, **wkbGeometryCollection** = 7,  
**wkbNone** = 100, **wkbLinearRing** = 101, **wkbPoint25D** = 0x80000001, **wkbLineString25D** = 0x80000002,  
**wkbPolygon25D** = 0x80000003, **wkbMultiPoint25D** = 0x80000004, **wkbMultiLineString25D** =  
0x80000005, **wkbMultiPolygon25D** = 0x80000006,  
**wkbGeometryCollection25D** = 0x80000007 }
- enum **OGRFieldType** {  
**OFTInteger** = 0, **OFTIntegerList** = 1, **OFTReal** = 2, **OFTRealList** = 3,  
**OFTString** = 4, **OFTStringList** = 5, **OFTWideString** = 6, **OFTWideStringList** = 7,  
**OFTBinary** = 8, **OFTDate** = 9, **OFTTime** = 10, **OFTDateTime** = 11 }
- enum **OGRJustification**
- enum **ogr\_style\_tool\_class\_id**
- enum **ogr\_style\_tool\_units\_id**
- enum **ogr\_style\_tool\_param\_pen\_id**
- enum **ogr\_style\_tool\_param\_brush\_id**
- enum **ogr\_style\_tool\_param\_symbol\_id**
- enum **ogr\_style\_tool\_param\_label\_id**

## Functions

- const char \* **OGRGeometryTypeToName** (**OGRwkbGeometryType** eType)  
*Fetch a human readable name corresponding to an OGRwkbGeometryType value. The returned value should not be modified, or freed by the application.*
- **OGRwkbGeometryType** **OGRMergeGeometryTypes** (**OGRwkbGeometryType** eMain, **OGRwkbGeometryType** eExtra)  
*Find common geometry type.*
- int **OGRParseDate** (const char \*pszInput, **OGRField** \*psOutput, int nOptions)
- int CPL\_STDCALL **GDALCheckVersion** (int nVersionMajor, int nVersionMinor, const char \*pszCallingComponentName)

### 13.13.1 Detailed Description

Core portability services for cross-platform OGR code.

### 13.13.2 Macro Definition Documentation

- 13.13.2.1 **#define** **GDAL\_CHECK\_VERSION**( *pszCallingComponentName* ) **GDALCheckVersion**(**GDAL\_VERSION\_MAJOR**, **GDAL\_VERSION\_MINOR**, *pszCallingComponentName*)

Helper macro for GDALCheckVersion

### 13.13.3 Typedef Documentation

- 13.13.3.1 **typedef** enum **ogr\_style\_tool\_param\_brush\_id** **OGRSTBrushParam**

List of parameters for use with **OGRStyleBrush** (p. ??).

- 13.13.3.2 **typedef** enum **ogr\_style\_tool\_class\_id** **OGRSTClassId**

**OGRStyleTool** (p. ??) derived class types (returned by GetType()).

## 13.13.3.3 typedef enum ogr\_style\_tool\_param\_label\_id OGRSTLabelParam

List of parameters for use with **OGRStyleLabel** (p. ??).

## 13.13.3.4 typedef enum ogr\_style\_tool\_param\_pen\_id OGRSTPenParam

List of parameters for use with **OGRStylePen** (p. ??).

## 13.13.3.5 typedef enum ogr\_style\_tool\_param\_symbol\_id OGRSTSymbolParam

List of parameters for use with **OGRStyleSymbol** (p. ??).

## 13.13.3.6 typedef enum ogr\_style\_tool\_units\_id OGRSTUnitId

List of units supported by OGRStyleTools.

## 13.13.4 Enumeration Type Documentation

## 13.13.4.1 enum ogr\_style\_tool\_class\_id

**OGRStyleTool** (p. ??) derived class types (returned by GetType()).

## 13.13.4.2 enum ogr\_style\_tool\_param\_brush\_id

List of parameters for use with **OGRStyleBrush** (p. ??).

## 13.13.4.3 enum ogr\_style\_tool\_param\_label\_id

List of parameters for use with **OGRStyleLabel** (p. ??).

## 13.13.4.4 enum ogr\_style\_tool\_param\_pen\_id

List of parameters for use with **OGRStylePen** (p. ??).

## 13.13.4.5 enum ogr\_style\_tool\_param\_symbol\_id

List of parameters for use with **OGRStyleSymbol** (p. ??).

## 13.13.4.6 enum ogr\_style\_tool\_units\_id

List of units supported by OGRStyleTools.

## 13.13.4.7 enum OGRFieldType

List of feature field types. This list is likely to be extended in the future ... avoid coding applications based on the assumption that all field types can be known.

## Enumerator

**OFTInteger** Simple 32bit integer

**OFTIntegerList** List of 32bit integers  
**OFTReal** Double Precision floating point  
**OFTRealList** List of doubles  
**OFTString** String of ASCII chars  
**OFTStringList** Array of strings  
**OFTWideString** deprecated  
**OFTWideStringList** deprecated  
**OFTBinary** Raw Binary data  
**OFTDate** Date  
**OFTTime** Time  
**OFTDateTime** Date and Time

#### 13.13.4.8 enum OGRJustification

Display justification for field values.

#### 13.13.4.9 enum OGRwkbGeometryType

List of well known binary geometry types. These are used within the BLOBs but are also returned from **OGRGeometry::getGeometryType()** (p. ??) to identify the type of a geometry object.

Enumerator

**wkbUnknown** unknown type, non-standard  
**wkbPoint** 0-dimensional geometric object, standard WKB  
**wkbLineString** 1-dimensional geometric object with linear interpolation between Points, standard WKB  
**wkbPolygon** planar 2-dimensional geometric object defined by 1 exterior boundary and 0 or more interior boundaries, standard WKB  
**wkbMultiPoint** GeometryCollection of Points, standard WKB  
**wkbMultiLineString** GeometryCollection of LineStrings, standard WKB  
**wkbMultiPolygon** GeometryCollection of Polygons, standard WKB  
**wkbGeometryCollection** geometric object that is a collection of 1 or more geometric objects, standard WKB  
**wkbNone** non-standard, for pure attribute records  
**wkbLinearRing** non-standard, just for createGeometry()  
**wkbPoint25D** 2.5D extension as per 99-402  
**wkbLineString25D** 2.5D extension as per 99-402  
**wkbPolygon25D** 2.5D extension as per 99-402  
**wkbMultiPoint25D** 2.5D extension as per 99-402  
**wkbMultiLineString25D** 2.5D extension as per 99-402  
**wkbMultiPolygon25D** 2.5D extension as per 99-402  
**wkbGeometryCollection25D** 2.5D extension as per 99-402

### 13.13.5 Function Documentation

13.13.5.1 int CPL\_STDCALL GDALCheckVersion ( int *nVersionMajor*, int *nVersionMinor*, const char \* *pszCallingComponentName* )

Return TRUE if GDAL library version at runtime matches *nVersionMajor*.*nVersionMinor*.

The purpose of this method is to ensure that calling code will run with the GDAL version it is compiled for. It is primarily intended for external plugins.



## Parameters

<i>nVersionMajor</i>	Major version to be tested against
<i>nVersionMinor</i>	Minor version to be tested against
<i>pszCalling-Component-Name</i>	If not NULL, in case of version mismatch, the method will issue a failure mentioning the name of the calling component.

13.13.5.2 `const char* OGRGeometryTypeToName ( OGRwkbGeometryType eType )`

Fetch a human readable name corresponding to an OGRwkbGeometryType value. The returned value should not be modified, or freed by the application.

This function is C callable.

## Parameters

<i>eType</i>	the geometry type.
--------------	--------------------

## Returns

internal human readable string, or NULL on failure.

References wkbGeometryCollection, wkbLineString, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, wkbNone, wkbPoint, wkbPolygon, and wkbUnknown.

13.13.5.3 `OGRwkbGeometryType OGRMergeGeometryTypes ( OGRwkbGeometryType eMain, OGRwkbGeometryType eExtra )`

Find common geometry type.

Given two geometry types, find the most specific common type. Normally used repeatedly with the geometries in a layer to try and establish the most specific geometry type that can be reported for the layer.

NOTE: wkbUnknown is the "worst case" indicating a mixture of geometry types with nothing in common but the base geometry type. wkbNone should be used to indicate that no geometries have been encountered yet, and means the first geometry encountered will establish the preliminary type.

## Parameters

<i>eMain</i>	the first input geometry type.
<i>eExtra</i>	the second input geometry type.

## Returns

the merged geometry type.

References wkbGeometryCollection, wkbMultiLineString, wkbMultiPoint, wkbMultiPolygon, wkbNone, and wkbUnknown.

Referenced by OGRUnionLayer::GetGeomType().

13.13.5.4 `int OGRParseDate ( const char * pszInput, OGRField * psField, int nOptions )`

Parse date string.

This function attempts to parse a date string in a variety of formats into the OGRField.Date format suitable for use with OGR. Generally speaking this function is expecting values like:

YYYY-MM-DD HH:MM:SS+nn

The seconds may also have a decimal portion (which is ignored). And just dates (YYYY-MM-DD) or just times (HH:MM:SS) are also supported. The date may also be in YYYY/MM/DD format. If the year is less than 100 and greater than 30 a "1900" century value will be set. If it is less than 30 and greater than -1 then a "2000" century value will be set. In the future this function may be generalized, and additional control provided through `nOptions`, but an `nOptions` value of "0" should always do a reasonable default form of processing.

The value of `psField` will be indeterminate if the function fails (returns FALSE).

#### Parameters

<i>pszInput</i>	the input date string.
<i>psField</i>	the <b>OGRField</b> (p. ??) that will be updated with the parsed result.
<i>nOptions</i>	parsing options, for now always 0.

#### Returns

TRUE if apparently successful or FALSE on failure.

References `CPLError()`, and `CPLScanLong()`.

Referenced by `OGRFeature::SetField()`.

## 13.14 ogr\_feature.h File Reference

```
#include "ogr_geometry.h"
#include "ogr_featurestyle.h"
#include "cpl_atomic_ops.h"
```

#### Classes

- class **OGRFieldDefn**
- class **OGRFeatureDefn**
- class **OGRFeature**
- class **OGRFeatureQuery**

### 13.14.1 Detailed Description

Simple feature classes.

## 13.15 ogr\_featurestyle.h File Reference

```
#include "cpl_conv.h"
#include "cpl_string.h"
#include "ogr_core.h"
```

#### Classes

- struct **ogr\_style\_param**
- struct **ogr\_style\_value**
- class **OGRStyleTable**
- class **OGRStyleMgr**

- class **OGRStyleTool**
- class **OGRStylePen**
- class **OGRStyleBrush**
- class **OGRStyleSymbol**
- class **OGRStyleLabel**

### 13.15.1 Detailed Description

Simple feature style classes.

## 13.16 ogr\_geocoding.h File Reference

```
#include "cpl_port.h"
#include "ogr_api.h"
```

### Functions

- **OGRGeocodingSessionH OGRGeocodeCreateSession** (char \*\*papszOptions)  
*Creates a session handle for geocoding requests.*
- void **OGRGeocodeDestroySession** (**OGRGeocodingSessionH** hSession)  
*Destroys a session handle for geocoding requests.*
- **OGRLayerH OGRGeocode** (**OGRGeocodingSessionH** hSession, const char \*pszQuery, char \*\*papszStructuredQuery, char \*\*papszOptions)  
*Runs a geocoding request.*
- **OGRLayerH OGRGeocodeReverse** (**OGRGeocodingSessionH** hSession, double dfLon, double dfLat, char \*\*papszOptions)  
*Runs a reverse geocoding request.*
- void **OGRGeocodeFreeResult** (**OGRLayerH** hLayer)  
*Destroys the result of a geocoding request.*

### 13.16.1 Detailed Description

C API for geocoding client.

### 13.16.2 Function Documentation

**13.16.2.1 OGRLayerH OGRGeocode** ( **OGRGeocodingSessionH** hSession, const char \* pszQuery, char \*\* papszStructuredQuery, char \*\* papszOptions )

Runs a geocoding request.

If the result is not found in cache, a GET request will be sent to resolve the query.

Note: most online services have Term of Uses. You are kindly requested to read and follow them. For the OpenStreetMap Nominatim service, this implementation will make sure that no more than one request is sent by second, but there might be other restrictions that you must follow by other means.

In case of success, the return of this function is a OGR layer that contain zero, one or several features matching the query. Note that the geometry of the features is not necessarily a point. The returned layer must be freed with **OGRGeocodeFreeResult()** (p. ??).

Note: this function is also available as the SQL `ogr_geocode()` function of the SQL SQLite dialect.

The list of recognized options is :

- ADDRESSDETAILS=0 or 1: Include a breakdown of the address into elements Defaults to 1. (Known to work with OSM and MapQuest Nominatim)
- COUNTRYCODES=code1,code2,...codeN: Limit search results to a specific country (or a list of countries). The codes must follow ISO 3166-1, i.e. gb for United Kingdom, de for Germany, etc.. (Known to work with OSM and MapQuest Nominatim)
- LIMIT=number: the number of records to return. Unlimited if not specified. (Known to work with OSM and MapQuest Nominatim)
- RAW\_FEATURE=YES: to specify that a 'raw' field must be added to the returned feature with the raw XML content.
- EXTRA\_QUERY\_PARAMETERS=params: additional parameters for the GET request.

#### Parameters

<i>hSession</i>	the geocoding session handle.
<i>pszQuery</i>	the string to geocode.
<i>papsz-StructuredQuery</i>	unused for now. Must be NULL.
<i>papszOptions</i>	a list of options or NULL.

#### Returns

a OGR layer with the result(s), or NULL in case of error. The returned layer must be freed with **OGRGeocodeFreeResult()** (p. ??).

#### Since

GDAL 1.10

References CPLError(), and CPLEscapeString().

#### 13.16.2.2 OGRGeocodingSessionH OGRGeocodeCreateSession ( char \*\* *papszOptions* )

Creates a session handle for geocoding requests.

Available *papszOptions* values:

- "CACHE\_FILE" : Defaults to "ogr\_geocode\_cache.sqlite" (or otherwise "ogr\_geocode\_cache.csv" if the SQLite driver isn't available). Might be any CSV, SQLite or PostgreSQL datasource.
- "READ\_CACHE" : "TRUE" (default) or "FALSE"
- "WRITE\_CACHE" : "TRUE" (default) or "FALSE"
- "SERVICE": "OSM\_NOMINATIM" (default), "MAPQUEST\_NOMINATIM", "YAHOO", "GEONAMES", "BING" or other value. Note: "YAHOO" is no longer available as a free service.
- "EMAIL": used by OSM\_NOMINATIM. Optional, but recommended.
- "USERNAME": used by GEONAMES. Compulsory in that case.
- "KEY": used by BING. Compulsory in that case.
- "APPLICATION": used to set the User-Agent MIME header. Defaults to GDAL/OGR version string.
- "LANGUAGE": used to set the Accept-Language MIME header. Preferred language order for showing search results.
- "DELAY": minimum delay, in second, between 2 consecutive queries. Defaults to 1.0.

- "QUERY\_TEMPLATE": URL template for GET requests. Must contain one and only one occurrence of %s in it. If not specified, for SERVICE=OSM\_NOMINATIM, MAPQUEST\_NOMINATIM, YAHOO, GEONAMES or BING, the URL template is hard-coded.
- "REVERSE\_QUERY\_TEMPLATE": URL template for GET requests for reverse geocoding. Must contain one and only one occurrence of {lon} and {lat} in it. If not specified, for SERVICE=OSM\_NOMINATIM, MAPQUEST\_NOMINATIM, YAHOO, GEONAMES or BING, the URL template is hard-coded.

All the above options can also be set by defining the configuration option of the same name, prefixed by OGR\_GEOCODE\_. For example "OGR\_GEOCODE\_SERVICE" for the "SERVICE" option.

#### Parameters

<i>papszOptions</i>	NULL, or a NULL-terminated list of string options.
---------------------	--

#### Returns

an handle that should be freed with **OGRGeocodeDestroySession()** (p. ??), or NULL in case of failure.

#### Since

GDAL 1.10

References CPLAtofM(), CPLCalloc(), CPLError(), CPLGetExtension(), CPLStrdup(), CSLTestBoolean(), and OGRGeocodeDestroySession().

#### 13.16.2.3 void OGRGeocodeDestroySession ( OGRGeocodingSessionH hSession )

Destroys a session handle for geocoding requests.

#### Parameters

<i>hSession</i>	the handle to destroy.
-----------------	------------------------

#### Since

GDAL 1.10

References OGRReleaseDataSource().

Referenced by OGRGeocodeCreateSession().

#### 13.16.2.4 void OGRGeocodeFreeResult ( OGRLayerH hLayer )

Destroys the result of a geocoding request.

#### Parameters

<i>hLayer</i>	the layer returned by <b>OGRGeocode()</b> (p. ??) or <b>OGRGeocodeReverse()</b> (p. ??) to destroy.
---------------	---

#### Since

GDAL 1.10

#### 13.16.2.5 OGRLayerH OGRGeocodeReverse ( OGRGeocodingSessionH hSession, double dfLon, double dfLat, char \*\* papszOptions )

Runs a reverse geocoding request.

If the result is not found in cache, a GET request will be sent to resolve the query.

Note: most online services have Term of Uses. You are kindly requested to read and follow them. For the Open-StreetMap Nominatim service, this implementation will make sure that no more than one request is sent by second, but there might be other restrictions that you must follow by other means.

In case of success, the return of this function is a OGR layer that contain zero, one or several features matching the query. The returned layer must be freed with **OGRGeocodeFreeResult()** (p. ??).

Note: this function is also available as the SQL `ogr_geocode_reverse()` function of the SQL SQLite dialect.

The list of recognized options is :

- **ZOOM=a\_level**: to query a specific zoom level. Only understood by the OSM Nominatim service.
- **RAW\_FEATURE=YES**: to specify that a 'raw' field must be added to the returned feature with the raw XML content.
- **EXTRA\_QUERY\_PARAMETERS=params**: additionnal parameters for the GET request for reverse geocoding.

#### Parameters

<i>hSession</i>	the geocoding session handle.
<i>dfLon</i>	the longitude.
<i>dfLat</i>	the latitude.
<i>papszOptions</i>	a list of options or NULL.

#### Returns

a OGR layer with the result(s), or NULL in case of error. The returned layer must be freed with **OGRGeocodeFreeResult()** (p. ??).

#### Since

GDAL 1.10

References `CPLError()`.

## 13.17 ogr\_geometry.h File Reference

```
#include "ogr_core.h"
#include "ogr_spatialref.h"
```

#### Classes

- class **OGRRawPoint**
- class **OGRGeometry**
- class **OGRPoint**
- class **OGRCurve**
- class **OGRLineString**
- class **OGRLinearRing**
- class **OGRSurface**
- class **OGRPolygon**
- class **OGRGeometryCollection**
- class **OGRMultiPolygon**
- class **OGRMultiPoint**
- class **OGRMultiLineString**
- class **OGRGeometryFactory**

### 13.17.1 Detailed Description

Simple feature geometry classes.

## 13.18 ogr\_spatialref.h File Reference

```
#include "ogr_srs_api.h"
```

### Classes

- class **OGR\_SRSNode**
- class **OGRSpatialReference**
- class **OGRCoordinateTransformation**

### Functions

- **OGRCoordinateTransformation \* OGRCreateCoordinateTransformation** (**OGRSpatialReference** \*poSource, **OGRSpatialReference** \*poTarget)

### 13.18.1 Detailed Description

Coordinate systems services.

### 13.18.2 Function Documentation

#### 13.18.2.1 **OGRCoordinateTransformation\* OGRCreateCoordinateTransformation** ( **OGRSpatialReference** \*poSource, **OGRSpatialReference** \* poTarget )

Create transformation object.

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

Input spatial reference system objects are assigned by copy (calling clone() method) and no ownership transfer occurs.

The delete operator, or **OCTDestroyCoordinateTransformation()** (p. ??) should be used to destroy transformation objects.

The PROJ.4 library must be available at run-time.

#### Parameters

<i>poSource</i>	source spatial reference system.
<i>poTarget</i>	target spatial reference system.

#### Returns

NULL on failure or a ready to use transformation object.

References CPLError().

Referenced by OCTNewCoordinateTransformation(), and OGRGeometry::transformTo().

## 13.19 ogr\_srs\_api.h File Reference

```
#include "ogr_core.h"
```

### Functions

- const char \* **OSRAxisEnumToName** (OGRAxisOrientation eOrientation)  
*Return the string representation for the OGRAxisOrientation enumeration.*
- OGRSpatialReferenceH CPL\_STDCALL **OSRNewSpatialReference** (const char \*)  
*Constructor.*
- OGRSpatialReferenceH CPL\_STDCALL **OSRCloneGeogCS** (OGRSpatialReferenceH)  
*Make a duplicate of the GEOGCS node of this **OGRSpatialReference** (p. ??) object.*
- OGRSpatialReferenceH CPL\_STDCALL **OSRClone** (OGRSpatialReferenceH)  
*Make a duplicate of this **OGRSpatialReference** (p. ??).*
- void CPL\_STDCALL **OSRDestroySpatialReference** (OGRSpatialReferenceH)  
***OGRSpatialReference** (p. ??) destructor.*
- int **OSRReference** (OGRSpatialReferenceH)  
*Increments the reference count by one.*
- int **OSRDereference** (OGRSpatialReferenceH)  
*Decrements the reference count by one.*
- void **OSRRelease** (OGRSpatialReferenceH)  
*Decrements the reference count by one, and destroy if zero.*
- OGRErr **OSRValidate** (OGRSpatialReferenceH)  
*Validate SRS tokens.*
- OGRErr **OSRFixupOrdering** (OGRSpatialReferenceH)  
*Correct parameter ordering to match CT Specification.*
- OGRErr **OSRFixup** (OGRSpatialReferenceH)  
*Fixup as needed.*
- OGRErr **OSRStripCTParms** (OGRSpatialReferenceH)  
*Strip OGC CT Parameters.*
- OGRErr CPL\_STDCALL **OSRImportFromEPSG** (OGRSpatialReferenceH, int)  
*Initialize SRS based on EPSG GCS or PCS code.*
- OGRErr CPL\_STDCALL **OSRImportFromEPSGA** (OGRSpatialReferenceH, int)  
*Initialize SRS based on EPSG GCS or PCS code.*
- OGRErr **OSRImportFromWkt** (OGRSpatialReferenceH, char \*\*)  
*Import from WKT string.*
- OGRErr **OSRImportFromProj4** (OGRSpatialReferenceH, const char \*)  
*Import PROJ.4 coordinate string.*
- OGRErr **OSRImportFromESRI** (OGRSpatialReferenceH, char \*\*)  
*Import coordinate system from ESRI .prj format(s).*
- OGRErr **OSRImportFromPCI** (OGRSpatialReferenceH hSRS, const char \*, const char \*, double \*)  
*Import coordinate system from PCI projection definition.*
- OGRErr **OSRImportFromUSGS** (OGRSpatialReferenceH, long, long, double \*, long)  
*Import coordinate system from USGS projection definition.*
- OGRErr **OSRImportFromXML** (OGRSpatialReferenceH, const char \*)  
*Import coordinate system from XML format (GML only currently).*
- OGRErr **OSRImportFromMICoordSys** (OGRSpatialReferenceH, const char \*)  
*Import Mapinfo style CoordSys definition.*
- OGRErr **OSRImportFromERM** (OGRSpatialReferenceH, const char \*, const char \*, const char \*)



- Create OGR WKT from ERMapper projection definitions.*

  - OGRErr **OSRImportFromUrl** (OGRSpatialReferenceH, const char \*)

*Set spatial reference from a URL.*
- OGRErr CPL\_STDCALL **OSRExportToWkt** (OGRSpatialReferenceH, char \*\*)

*Convert this SRS into WKT format.*
- OGRErr CPL\_STDCALL **OSRExportToPrettyWkt** (OGRSpatialReferenceH, char \*\*, int)

*Convert this SRS into a nicely formatted WKT string for display to a person.*
- OGRErr CPL\_STDCALL **OSRExportToProj4** (OGRSpatialReferenceH, char \*\*)

*Export coordinate system in PROJ.4 format.*
- OGRErr **OSRExportToPCI** (OGRSpatialReferenceH, char \*\*, char \*\*, double \*\*)

*Export coordinate system in PCI projection definition.*
- OGRErr **OSRExportToUSGS** (OGRSpatialReferenceH, long \*, long \*, double \*\*, long \*)

*Export coordinate system in USGS GCTP projection definition.*
- OGRErr **OSRExportToXML** (OGRSpatialReferenceH, char \*\*, const char \*)

*Export coordinate system in XML format.*
- OGRErr **OSRExportToMICoordSys** (OGRSpatialReferenceH, char \*\*)

*Export coordinate system in Mapinfo style CoordSys format.*
- OGRErr **OSRExportToERM** (OGRSpatialReferenceH, char \*, char \*, char \*)

*Convert coordinate system to ERMapper format.*
- OGRErr **OSRMorphToESRI** (OGRSpatialReferenceH)

*Convert in place to ESRI WKT format.*
- OGRErr **OSRMorphFromESRI** (OGRSpatialReferenceH)

*Convert in place from ESRI WKT format.*
- OGRErr CPL\_STDCALL **OSRSetAttrValue** (OGRSpatialReferenceH hSRS, const char \*pszNodePath, const char \*pszNewNodeValue)

*Set attribute value in spatial reference.*
- const char \*CPL\_STDCALL **OSRGetAttrValue** (OGRSpatialReferenceH hSRS, const char \*pszName, int iChild)

*Fetch indicated attribute of named node.*
- OGRErr **OSRSetAngularUnits** (OGRSpatialReferenceH, const char \*, double)

*Set the angular units for the geographic coordinate system.*
- double **OSRGetAngularUnits** (OGRSpatialReferenceH, char \*\*)

*Fetch angular geographic coordinate system units.*
- OGRErr **OSRSetLinearUnits** (OGRSpatialReferenceH, const char \*, double)

*Set the linear units for the projection.*
- OGRErr **OSRSetTargetLinearUnits** (OGRSpatialReferenceH, const char \*, const char \*, double)

*Set the linear units for the target node.*
- OGRErr **OSRSetLinearUnitsAndUpdateParameters** (OGRSpatialReferenceH, const char \*, double)

*Set the linear units for the projection.*
- double **OSRGetLinearUnits** (OGRSpatialReferenceH, char \*\*)

*Fetch linear projection units.*
- double **OSRGetTargetLinearUnits** (OGRSpatialReferenceH, const char \*, char \*\*)

*Fetch linear projection units.*
- double **OSRGetPrimeMeridian** (OGRSpatialReferenceH, char \*\*)

*Fetch prime meridian info.*
- int **OSRIsGeographic** (OGRSpatialReferenceH)

*Check if geographic coordinate system.*
- int **OSRIsLocal** (OGRSpatialReferenceH)

*Check if local coordinate system.*
- int **OSRIsProjected** (OGRSpatialReferenceH)

*Check if projected coordinate system.*

- int **OSRIsCompound** (OGRSpatialReferenceH)  
*Check if the coordinate system is compound.*
- int **OSRIsGeocentric** (OGRSpatialReferenceH)  
*Check if geocentric coordinate system.*
- int **OSRIsVertical** (OGRSpatialReferenceH)  
*Check if vertical coordinate system.*
- int **OSRIsSameGeogCS** (OGRSpatialReferenceH, OGRSpatialReferenceH)  
*Do the GeogCS'es match?*
- int **OSRIsSameVertCS** (OGRSpatialReferenceH, OGRSpatialReferenceH)  
*Do the VertCS'es match?*
- int **OSRIsSame** (OGRSpatialReferenceH, OGRSpatialReferenceH)  
*Do these two spatial references describe the same system ?*
- OGRErr **OSRSetLocalCS** (OGRSpatialReferenceH hSRS, const char \*pszName)  
*Set the user visible LOCAL\_CS name.*
- OGRErr **OSRSetProjCS** (OGRSpatialReferenceH hSRS, const char \*pszName)  
*Set the user visible PROJCS name.*
- OGRErr **OSRSetGeocCS** (OGRSpatialReferenceH hSRS, const char \*pszName)  
*Set the user visible PROJCS name.*
- OGRErr **OSRSetWellKnownGeogCS** (OGRSpatialReferenceH hSRS, const char \*pszName)  
*Set a GeogCS based on well known name.*
- OGRErr CPL\_STDCALL **OSRSetFromUserInput** (OGRSpatialReferenceH hSRS, const char \*)  
*Set spatial reference from various text formats.*
- OGRErr **OSRCopyGeogCSFrom** (OGRSpatialReferenceH hSRS, OGRSpatialReferenceH hSrcSRS)  
*Copy GEOGCS from another **OGRSpatialReference** (p. ??).*
- OGRErr **OSRSetTOWGS84** (OGRSpatialReferenceH hSRS, double, double, double, double, double, double, double)  
*Set the Bursa-Wolf conversion to WGS84.*
- OGRErr **OSRGetTOWGS84** (OGRSpatialReferenceH hSRS, double \*, int)  
*Fetch TOWGS84 parameters, if available.*
- OGRErr **OSRSetCompoundCS** (OGRSpatialReferenceH hSRS, const char \*pszName, OGRSpatialReferenceH hHorizSRS, OGRSpatialReferenceH hVertSRS)  
*Setup a compound coordinate system.*
- OGRErr **OSRSetGeogCS** (OGRSpatialReferenceH hSRS, const char \*pszGeogName, const char \*pszDatumName, const char \*pszEllipsoidName, double dfSemiMajor, double dfInvFlattening, const char \*pszPMName, double dfPMOffset, const char \*pszUnits, double dfConvertToRadians)  
*Set geographic coordinate system.*
- OGRErr **OSRSetVertCS** (OGRSpatialReferenceH hSRS, const char \*pszVertCSName, const char \*pszVertDatumName, int nVertDatumType)  
*Setup the vertical coordinate system.*
- double **OSRGetSemiMajor** (OGRSpatialReferenceH, OGRErr \*)  
*Get spheroid semi major axis.*
- double **OSRGetSemiMinor** (OGRSpatialReferenceH, OGRErr \*)  
*Get spheroid semi minor axis.*
- double **OSRGetInvFlattening** (OGRSpatialReferenceH, OGRErr \*)  
*Get spheroid inverse flattening.*
- OGRErr **OSRSetAuthority** (OGRSpatialReferenceH hSRS, const char \*pszTargetKey, const char \*pszAuthority, int nCode)  
*Set the authority for a node.*
- const char \* **OSRGetAuthorityCode** (OGRSpatialReferenceH hSRS, const char \*pszTargetKey)  
*Get the authority code for a node.*
- const char \* **OSRGetAuthorityName** (OGRSpatialReferenceH hSRS, const char \*pszTargetKey)

- Get the authority name for a node.*

  - OGRErr **OSRSetProjection** (OGRSpatialReferenceH, const char \*)

*Set a projection name.*

  - OGRErr **OSRSetProjParm** (OGRSpatialReferenceH, const char \*, double)

*Set a projection parameter value.*

  - double **OSRGetProjParm** (OGRSpatialReferenceH hSRS, const char \*pszParmName, double dfDefault, OGRErr \*)

*Fetch a projection parameter value.*

  - OGRErr **OSRSetNormProjParm** (OGRSpatialReferenceH, const char \*, double)

*Set a projection parameter with a normalized value.*

  - double **OSRGetNormProjParm** (OGRSpatialReferenceH hSRS, const char \*pszParmName, double dfDefault, OGRErr \*)

*This function is the same as **OGRSpatialReference** (p. ??):*

  - OGRErr **OSRSetUTM** (OGRSpatialReferenceH hSRS, int nZone, int bNorth)

*Set UTM projection definition.*

  - int **OSRGetUTMZone** (OGRSpatialReferenceH hSRS, int \*pbNorth)

*Get utm zone information.*

  - OGRErr **OSRSetStatePlane** (OGRSpatialReferenceH hSRS, int nZone, int bNAD83)

*Set State Plane projection definition.*

  - OGRErr **OSRSetStatePlaneWithUnits** (OGRSpatialReferenceH hSRS, int nZone, int bNAD83, const char \*pszOverrideUnitName, double dfOverrideUnit)

*Set State Plane projection definition.*

  - OGRErr **OSRAutoidentifyEPSG** (OGRSpatialReferenceH hSRS)

*Set EPSG authority info if possible.*

  - int **OSREPSGTreatsAsLatLong** (OGRSpatialReferenceH hSRS)

*This function returns TRUE if EPSG feels this geographic coordinate system should be treated as having lat/long coordinate ordering.*

  - int **OSREPSGTreatsAsNorthingEasting** (OGRSpatialReferenceH hSRS)

*This function returns TRUE if EPSG feels this geographic coordinate system should be treated as having northing/easting coordinate ordering.*

  - const char \* **OSRGetAxis** (OGRSpatialReferenceH hSRS, const char \*pszTargetKey, int iAxis, OGRAxisOrientation \*peOrientation)

*Fetch the orientation of one axis.*

  - OGRErr **OSRSetACEA** (OGRSpatialReferenceH hSRS, double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
  - OGRErr **OSRSetAE** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
  - OGRErr **OSRSetBonne** (OGRSpatialReferenceH hSRS, double dfStandardParallel, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
  - OGRErr **OSRSetCEA** (OGRSpatialReferenceH hSRS, double dfStdP1, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
  - OGRErr **OSRSetCS** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
  - OGRErr **OSRSetEC** (OGRSpatialReferenceH hSRS, double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
  - OGRErr **OSRSetEckert** (OGRSpatialReferenceH hSRS, int nVariation, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
  - OGRErr **OSRSetEckertIV** (OGRSpatialReferenceH hSRS, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
  - OGRErr **OSRSetEckertVI** (OGRSpatialReferenceH hSRS, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
  - OGRErr **OSRSetEquiarectangular** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)

- OGRErr **OSRSetEquirectangular2** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfPseudoStdParallel1, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetGS** (OGRSpatialReferenceH hSRS, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetGH** (OGRSpatialReferenceH hSRS, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetIGH** (OGRSpatialReferenceH hSRS)
- OGRErr **OSRSetGEOS** (OGRSpatialReferenceH hSRS, double dfCentralMeridian, double dfSatelliteHeight, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetGaussSchreiberTMercator** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetGnomonic** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetOM** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfAzimuth, double dfRectToSkew, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetHOM** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfAzimuth, double dfRectToSkew, double dfScale, double dfFalseEasting, double dfFalseNorthing)

*Set a Hotine Oblique Mercator projection using azimuth angle.*

- OGRErr **OSRSetHOM2PNO** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfLat1, double dfLong1, double dfLat2, double dfLong2, double dfScale, double dfFalseEasting, double dfFalseNorthing)

*Set a Hotine Oblique Mercator projection using two points on projection centerline.*

- OGRErr **OSRSetIWMPolyconic** (OGRSpatialReferenceH hSRS, double dfLat1, double dfLat2, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetKrovak** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfAzimuth, double dfPseudoStdParallelLat, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetLAEA** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetLCC** (OGRSpatialReferenceH hSRS, double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetLCC1SP** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetLCCB** (OGRSpatialReferenceH hSRS, double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetMC** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetMercator** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetMollweide** (OGRSpatialReferenceH hSRS, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetNZMG** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetOS** (OGRSpatialReferenceH hSRS, double dfOriginLat, double dfCMeridian, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetOrthographic** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetPolyconic** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetPS** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetRobinson** (OGRSpatialReferenceH hSRS, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetSinusoidal** (OGRSpatialReferenceH hSRS, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetStereographic** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)

- OGRErr **OSRSetSOC** (OGRSpatialReferenceH hSRS, double dfLatitudeOfOrigin, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetTM** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetTMVariant** (OGRSpatialReferenceH hSRS, const char \*pszVariantName, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetTMG** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetTMSO** (OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetVDG** (OGRSpatialReferenceH hSRS, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing)
- OGRErr **OSRSetWagner** (OGRSpatialReferenceH hSRS, int nVariation, double dfFalseEasting, double dfFalseNorthing)
- void **OSRCleanup** (void)  
*Cleanup cached SRS related memory.*
- OGRCoordinateTransformationH  
CPL\_STDCALL **OCTNewCoordinateTransformation** (OGRSpatialReferenceH hSourceSRS, OGRSpatialReferenceH hTargetSRS)
- void CPL\_STDCALL **OCTDestroyCoordinateTransformation** (OGRCoordinateTransformationH)  
*OGRCoordinateTransformation (p. ??) destructor.*
- char \*\* **OPTGetProjectionMethods** ()
- char \*\* **OPTGetParameterList** (const char \*pszProjectionMethod, char \*\*ppszUserName)
- int **OPTGetParameterInfo** (const char \*pszProjectionMethod, const char \*pszParameterName, char \*\*ppszUserName, char \*\*ppszType, double \*pdfDefaultValue)

### 13.19.1 Detailed Description

C spatial reference system services and defines.

See also: **ogr\_spatialref.h** (p. ??)

### 13.19.2 Function Documentation

#### 13.19.2.1 void CPL\_STDCALL OCTDestroyCoordinateTransformation ( OGRCoordinateTransformationH hCT )

**OGRCoordinateTransformation** (p. ??) destructor.

This function is the same as **OGRCoordinateTransformation::DestroyCT()** (p. ??)

Parameters

<i>hCT</i>	the object to delete
------------	----------------------

#### 13.19.2.2 OGRCoordinateTransformationH CPL\_STDCALL OCTNewCoordinateTransformation ( OGRSpatialReferenceH hSourceSRS, OGRSpatialReferenceH hTargetSRS )

Create transformation object.

This is the same as the C++ function **OGRCreateCoordinateTransformation()** (p. ??).

Input spatial reference system objects are assigned by copy (calling clone() method) and no ownership transfer occurs.

**OCTDestroyCoordinateTransformation()** (p. ??) should be used to destroy transformation objects.

The PROJ.4 library must be available at run-time.

## Parameters

<i>hSourceSRS</i>	source spatial reference system.
<i>hTargetSRS</i>	target spatial reference system.

## Returns

NULL on failure or a ready to use transformation object.

References OGRCreateCoordinateTransformation().

**13.19.2.3** `int OPTGetParameterInfo ( const char * pszProjectionMethod, const char * pszParameterName, char ** ppszUserName, char ** ppszType, double * pdfDefaultValue )`

Fetch information about a single parameter of a projection method.

## Parameters

<i>pszProjection-Method</i>	name of projection method for which the parameter applies. Not currently used, but in the future this could affect defaults. This is the internal projection method name, such as "-Transverse_Mercator".
<i>pszParameter-Name</i>	name of the parameter to fetch information about. This is the internal name such as "central_meridian" (SRS_PP_CENTRAL_MERIDIAN).
<i>ppszUserName</i>	location at which to return the user visible name for the parameter. This pointer may be NULL to skip the user name. The returned name should not be modified or freed.
<i>ppszType</i>	location at which to return the parameter type for the parameter. This pointer may be NULL to skip. The returned type should not be modified or freed. The type values are described above.
<i>pdfDefaultValue</i>	location at which to put the default value for this parameter. The pointer may be NULL.

## Returns

TRUE if parameter found, or FALSE otherwise.

**13.19.2.4** `char** OPTGetParameterList ( const char * pszProjectionMethod, char ** ppszUserName )`

Fetch the parameters for a given projection method.

## Parameters

<i>pszProjection-Method</i>	internal name of projection methods to fetch the parameters for, such as "Transverse_Mercator" (SRS_PT_TRANSVERSE_MERCATOR).
<i>ppszUserName</i>	pointer in which to return a user visible name for the projection name. The returned string should not be modified or freed by the caller. Legal to pass in NULL if user name not required.

## Returns

returns a NULL terminated list of internal parameter names that should be freed by the caller when no longer needed. Returns NULL if projection method is unknown.

References CPLCalloc().

**13.19.2.5** `char** OPTGetProjectionMethods ( )`

Fetch list of possible projection methods.

**Returns**

Returns NULL terminated list of projection methods. This should be freed with **CSLDestroy()** (p. ??) when no longer needed.

**13.19.2.6 OGRErr OSRAutoIdentifyEPSG ( OGRSpatialReferenceH hSRS )**

Set EPSG authority info if possible.

This function is the same as **OGRSpatialReference::AutoIdentifyEPSG()** (p. ??).

**13.19.2.7 const char\* OSRAxisEnumToName ( OGRAxisOrientation eOrientation )**

Return the string representation for the OGRAxisOrientation enumeration.

For example "NORTH" for OAO\_North.

**Returns**

an internal string

Referenced by **OGRSpatialReference::SetAxes()**.

**13.19.2.8 void OSRCleanup ( void )**

Cleanup cached SRS related memory.

This function will attempt to cleanup any cache spatial reference related information, such as cached tables of coordinate systems.

Referenced by **OGRCleanupAll()**.

**13.19.2.9 OGRSpatialReferenceH CPL\_STDCALL OSRClone ( OGRSpatialReferenceH hSRS )**

Make a duplicate of this **OGRSpatialReference** (p. ??).

This function is the same as **OGRSpatialReference::Clone()** (p. ??)

**13.19.2.10 OGRSpatialReferenceH CPL\_STDCALL OSRCloneGeogCS ( OGRSpatialReferenceH hSource )**

Make a duplicate of the GEOGCS node of this **OGRSpatialReference** (p. ??) object.

This function is the same as **OGRSpatialReference::CloneGeogCS()** (p. ??).

**13.19.2.11 OGRErr OSRCopyGeogCSFrom ( OGRSpatialReferenceH hSRS, OGRSpatialReferenceH hSrcSRS )**

Copy GEOGCS from another **OGRSpatialReference** (p. ??).

This function is the same as **OGRSpatialReference::CopyGeogCSFrom()** (p. ??)

**13.19.2.12 int OSRDereference ( OGRSpatialReferenceH hSRS )**

Decrements the reference count by one.

This function is the same as **OGRSpatialReference::Dereference()** (p. ??)

13.19.2.13 void CPL\_STDCALL OSRDestroySpatialReference ( OGRSpatialReferenceH *hSRS* )

**OGRSpatialReference** (p. ??) destructor.

This function is the same as **OGRSpatialReference::~~OGRSpatialReference()** (p. ??) and **OGRSpatialReference::DestroySpatialReference()** (p. ??)

Parameters

<i>hSRS</i>	the object to delete
-------------	----------------------

13.19.2.14 int OSREPSGTreatsAsLatLong ( OGRSpatialReferenceH *hSRS* )

This function returns TRUE if EPSG feels this geographic coordinate system should be treated as having lat/long coordinate ordering.

This function is the same as **OGRSpatialReference::OSREPSGTreatsAsLatLong()**.

13.19.2.15 int OSREPSGTreatsAsNorthingEasting ( OGRSpatialReferenceH *hSRS* )

This function returns TRUE if EPSG feels this geographic coordinate system should be treated as having northing/easting coordinate ordering.

This function is the same as **OGRSpatialReference::EPSGTreatsAsNorthingEasting()** (p. ??).

Since

OGR 1.10.0

13.19.2.16 OGRErr OSRExportToERM ( OGRSpatialReferenceH *hSRS*, char \* *pszProj*, char \* *pszDatum*, char \* *pszUnits* )

Convert coordinate system to ERMapper format.

This function is the same as **OGRSpatialReference::exportToERM()** (p. ??).

13.19.2.17 OGRErr OSRExportToMICoordSys ( OGRSpatialReferenceH *hSRS*, char \*\* *ppszReturn* )

Export coordinate system in Mapinfo style CoordSys format.

This method is the equivalent of the C++ method **OGRSpatialReference::exportToMICoordSys** (p. ??)

13.19.2.18 OGRErr OSRExportToPCI ( OGRSpatialReferenceH *hSRS*, char \*\* *ppszProj*, char \*\* *ppszUnits*, double \*\* *ppadfPrjParams* )

Export coordinate system in PCI projection definition.

This function is the same as **OGRSpatialReference::exportToPCI()** (p. ??).

13.19.2.19 OGRErr CPL\_STDCALL OSRExportToPrettyWkt ( OGRSpatialReferenceH *hSRS*, char \*\* *ppszReturn*, int *bSimplify* )

Convert this SRS into a a nicely formatted WKT string for display to a person.

This function is the same as **OGRSpatialReference::exportToPrettyWkt()** (p. ??).



13.19.2.20 OGRErr CPL\_STDCALL OSRExportToProj4 ( OGRSpatialReferenceH *hSRS*, char \*\* *ppszReturn* )

Export coordinate system in PROJ.4 format.

This function is the same as **OGRSpatialReference::exportToProj4()** (p. ??).

13.19.2.21 OGRErr OSRExportToUSGS ( OGRSpatialReferenceH *hSRS*, long \* *piProjSys*, long \* *piZone*, double \*\* *ppadfPrjParams*, long \* *piDatum* )

Export coordinate system in USGS GCTP projection definition.

This function is the same as **OGRSpatialReference::exportToUSGS()** (p. ??).

13.19.2.22 OGRErr CPL\_STDCALL OSRExportToWkt ( OGRSpatialReferenceH *hSRS*, char \*\* *ppszReturn* )

Convert this SRS into WKT format.

This function is the same as **OGRSpatialReference::exportToWkt()** (p. ??).

13.19.2.23 OGRErr OSRExportToXML ( OGRSpatialReferenceH *hSRS*, char \*\* *ppszRawXML*, const char \* *pszDialect* )

Export coordinate system in XML format.

This function is the same as **OGRSpatialReference::exportToXML()** (p. ??).

13.19.2.24 OGRErr OSRFixup ( OGRSpatialReferenceH *hSRS* )

Fixup as needed.

This function is the same as **OGRSpatialReference::Fixup()** (p. ??).

13.19.2.25 OGRErr OSRFixupOrdering ( OGRSpatialReferenceH *hSRS* )

Correct parameter ordering to match CT Specification.

This function is the same as **OGRSpatialReference::FixupOrdering()** (p. ??).

13.19.2.26 double OSRGetAngularUnits ( OGRSpatialReferenceH *hSRS*, char \*\* *ppszName* )

Fetch angular geographic coordinate system units.

This function is the same as **OGRSpatialReference::GetAngularUnits()** (p. ??)

13.19.2.27 const char\* CPL\_STDCALL OSRGetAttrValue ( OGRSpatialReferenceH *hSRS*, const char \* *pszKey*, int *iChild* )

Fetch indicated attribute of named node.

This function is the same as **OGRSpatialReference::GetAttrValue()** (p. ??)

13.19.2.28 const char\* OSRGetAuthorityCode ( OGRSpatialReferenceH *hSRS*, const char \* *pszTargetKey* )

Get the authority code for a node.

This function is the same as **OGRSpatialReference::GetAuthorityCode()** (p. ??).

13.19.2.29 `const char* OSRGetAuthorityName ( OGRSpatialReferenceH hSRS, const char * pszTargetKey )`

Get the authority name for a node.

This function is the same as **OGRSpatialReference::GetAuthorityName()** (p. ??).

13.19.2.30 `const char* OSRGetAxis ( OGRSpatialReferenceH hSRS, const char * pszTargetKey, int iAxis, OGRAxisOrientation * peOrientation )`

Fetch the orientation of one axis.

This method is the equivalent of the C++ method **OGRSpatialReference::GetAxis** (p. ??)

13.19.2.31 `double OSRGetInvFlattening ( OGRSpatialReferenceH hSRS, OGRErr * pnErr )`

Get spheroid inverse flattening.

This function is the same as **OGRSpatialReference::GetInvFlattening()** (p. ??)

13.19.2.32 `double OSRGetLinearUnits ( OGRSpatialReferenceH hSRS, char ** ppszName )`

Fetch linear projection units.

This function is the same as **OGRSpatialReference::GetLinearUnits()** (p. ??)

13.19.2.33 `double OSRGetNormProjParm ( OGRSpatialReferenceH hSRS, const char * pszName, double dfDefaultValue, OGRErr * pnErr )`

This function is the same as **OGRSpatialReference** (p. ??)::

This function is the same as **OGRSpatialReference::GetNormProjParm()** (p. ??)

13.19.2.34 `double OSRGetPrimeMeridian ( OGRSpatialReferenceH hSRS, char ** ppszName )`

Fetch prime meridian info.

This function is the same as **OGRSpatialReference::GetPrimeMeridian()** (p. ??)

13.19.2.35 `double OSRGetProjParm ( OGRSpatialReferenceH hSRS, const char * pszName, double dfDefaultValue, OGRErr * pnErr )`

Fetch a projection parameter value.

This function is the same as **OGRSpatialReference::GetProjParm()** (p. ??)

13.19.2.36 `double OSRGetSemiMajor ( OGRSpatialReferenceH hSRS, OGRErr * pnErr )`

Get spheroid semi major axis.

This function is the same as **OGRSpatialReference::GetSemiMajor()** (p. ??)

13.19.2.37 `double OSRGetSemiMinor ( OGRSpatialReferenceH hSRS, OGRErr * pnErr )`

Get spheroid semi minor axis.

This function is the same as **OGRSpatialReference::GetSemiMinor()** (p. ??)

13.19.2.38 `double OSRGetTargetLinearUnits ( OGRSpatialReferenceH hSRS, const char * pszTargetKey, char ** ppszName )`

Fetch linear projection units.

This function is the same as **OGRSpatialReference::GetTargetLinearUnits()** (p. ??)

Since

OGR 1.9.0

13.19.2.39 `OGRERR OSRGetTOWGS84 ( OGRSpatialReferenceH hSRS, double * padfCoeff, int nCoeffCount )`

Fetch TOWGS84 parameters, if available.

This function is the same as **OGRSpatialReference::GetTOWGS84()** (p. ??).

13.19.2.40 `int OSRGetUTMZone ( OGRSpatialReferenceH hSRS, int * pbNorth )`

Get utm zone information.

This is the same as the C++ method **OGRSpatialReference::GetUTMZone()** (p. ??)

13.19.2.41 `OGRERR CPL_STDCALL OSRImportFromEPSG ( OGRSpatialReferenceH hSRS, int nCode )`

Initialize SRS based on EPSG GCS or PCS code.

This function is the same as **OGRSpatialReference::importFromEPSG()** (p. ??).

13.19.2.42 `OGRERR CPL_STDCALL OSRImportFromEPSGA ( OGRSpatialReferenceH hSRS, int nCode )`

Initialize SRS based on EPSG GCS or PCS code.

This function is the same as **OGRSpatialReference::importFromEPSGA()** (p. ??).

13.19.2.43 `OGRERR OSRImportFromERM ( OGRSpatialReferenceH hSRS, const char * pszProj, const char * pszDatum, const char * pszUnits )`

Create OGR WKT from ERMapper projection definitions.

This function is the same as **OGRSpatialReference::importFromERM()** (p. ??).

13.19.2.44 `OGRERR OSRImportFromESRI ( OGRSpatialReferenceH hSRS, char ** papszPrj )`

Import coordinate system from ESRI .prj format(s).

This function is the same as the C++ method **OGRSpatialReference::importFromESRI()** (p. ??)

13.19.2.45 `OGRERR OSRImportFromMICoordSys ( OGRSpatialReferenceH hSRS, const char * pszCoordSys )`

Import Mapinfo style CoordSys definition.

This method is the equivalent of the C++ method **OGRSpatialReference::importFromMICoordSys** (p. ??)

**13.19.2.46** `OGRErr OSRImportFromPCI ( OGRSpatialReferenceH hSRS, const char * pszProj, const char * pszUnits, double * padfPrjParams )`

Import coordinate system from PCI projection definition.

This function is the same as **OGRSpatialReference::importFromPCI()** (p. ??).

**13.19.2.47** `OGRErr OSRImportFromProj4 ( OGRSpatialReferenceH hSRS, const char * pszProj4 )`

Import PROJ.4 coordinate string.

This function is the same as **OGRSpatialReference::importFromProj4()** (p. ??).

**13.19.2.48** `OGRErr OSRImportFromUrl ( OGRSpatialReferenceH hSRS, const char * pszUrl )`

Set spatial reference from a URL.

This function is the same as **OGRSpatialReference::importFromUrl()** (p. ??)

**13.19.2.49** `OGRErr OSRImportFromUSGS ( OGRSpatialReferenceH hSRS, long iProjsys, long iZone, double * padfPrjParams, long iDatum )`

Import coordinate system from USGS projection definition.

This function is the same as **OGRSpatialReference::importFromUSGS()** (p. ??).

**13.19.2.50** `OGRErr OSRImportFromWkt ( OGRSpatialReferenceH hSRS, char ** ppszInput )`

Import from WKT string.

This function is the same as **OGRSpatialReference::importFromWkt()** (p. ??).

**13.19.2.51** `OGRErr OSRImportFromXML ( OGRSpatialReferenceH hSRS, const char * pszXML )`

Import coordinate system from XML format (GML only currently).

This function is the same as **OGRSpatialReference::importFromXML()** (p. ??).

**13.19.2.52** `int OSRIsCompound ( OGRSpatialReferenceH hSRS )`

Check if the coordinate system is compound.

This function is the same as **OGRSpatialReference::IsCompound()** (p. ??).

**13.19.2.53** `int OSRIsGeocentric ( OGRSpatialReferenceH hSRS )`

Check if geocentric coordinate system.

This function is the same as **OGRSpatialReference::IsGeocentric()** (p. ??).

Since

OGR 1.9.0

**13.19.2.54** int OSRIsGeographic ( OGRSpatialReferenceH *hSRS* )

Check if geographic coordinate system.

This function is the same as **OGRSpatialReference::IsGeographic()** (p. ??).

**13.19.2.55** int OSRIsLocal ( OGRSpatialReferenceH *hSRS* )

Check if local coordinate system.

This function is the same as **OGRSpatialReference::IsLocal()** (p. ??).

**13.19.2.56** int OSRIsProjected ( OGRSpatialReferenceH *hSRS* )

Check if projected coordinate system.

This function is the same as **OGRSpatialReference::IsProjected()** (p. ??).

**13.19.2.57** int OSRIsSame ( OGRSpatialReferenceH *hSRS1*, OGRSpatialReferenceH *hSRS2* )

Do these two spatial references describe the same system ?

This function is the same as **OGRSpatialReference::IsSame()** (p. ??).

**13.19.2.58** int OSRIsSameGeogCS ( OGRSpatialReferenceH *hSRS1*, OGRSpatialReferenceH *hSRS2* )

Do the GeogCS'es match?

This function is the same as **OGRSpatialReference::IsSameGeogCS()** (p. ??).

**13.19.2.59** int OSRIsSameVertCS ( OGRSpatialReferenceH *hSRS1*, OGRSpatialReferenceH *hSRS2* )

Do the VertCS'es match?

This function is the same as **OGRSpatialReference::IsSameVertCS()** (p. ??).

**13.19.2.60** int OSRIsVertical ( OGRSpatialReferenceH *hSRS* )

Check if vertical coordinate system.

This function is the same as **OGRSpatialReference::IsVertical()** (p. ??).

Since

OGR 1.8.0

**13.19.2.61** OGRErr OSRMorphFromESRI ( OGRSpatialReferenceH *hSRS* )

Convert in place from ESRI WKT format.

This function is the same as the C++ method **OGRSpatialReference::morphFromESRI()** (p. ??)

**13.19.2.62** OGRErr OSRMorphToESRI ( OGRSpatialReferenceH *hSRS* )

Convert in place to ESRI WKT format.

This function is the same as the C++ method **OGRSpatialReference::morphToESRI()** (p. ??)

13.19.2.63 **OGRSpatialReferenceH CPL\_STDCALL OSRNewSpatialReference ( const char \* *pszWKT* )**

Constructor.

This function is the same as **OGRSpatialReference::OGRSpatialReference()**

References **OGRSpatialReference::importFromWkt()**.

13.19.2.64 **int OSRReference ( OGRSpatialReferenceH *hSRS* )**

Increments the reference count by one.

This function is the same as **OGRSpatialReference::Reference()** (p. ??)

13.19.2.65 **void OSRRelease ( OGRSpatialReferenceH *hSRS* )**

Decrements the reference count by one, and destroy if zero.

This function is the same as **OGRSpatialReference::Release()** (p. ??)

13.19.2.66 **OGRERR OSRSetACEA ( OGRSpatialReferenceH *hSRS*, double *dfStdP1*, double *dfStdP2*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing* )**

Albers Conic Equal Area

13.19.2.67 **OGRERR OSRSetAE ( OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing* )**

Azimuthal Equidistant

13.19.2.68 **OGRERR OSRSetAngularUnits ( OGRSpatialReferenceH *hSRS*, const char \* *pszUnits*, double *dfInRadians* )**

Set the angular units for the geographic coordinate system.

This function is the same as **OGRSpatialReference::SetAngularUnits()** (p. ??)

13.19.2.69 **OGRERR CPL\_STDCALL OSRSetAttrValue ( OGRSpatialReferenceH *hSRS*, const char \* *pszPath*, const char \* *pszValue* )**

Set attribute value in spatial reference.

This function is the same as **OGRSpatialReference::SetNode()** (p. ??)

13.19.2.70 **OGRERR OSRSetAuthority ( OGRSpatialReferenceH *hSRS*, const char \* *pszTargetKey*, const char \* *pszAuthority*, int *nCode* )**

Set the authority for a node.

This function is the same as **OGRSpatialReference::SetAuthority()** (p. ??).

13.19.2.71 **OGRERR OSRSetBonne ( OGRSpatialReferenceH *hSRS*, double *dfStandardParallel*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing* )**

Bonne

13.19.2.72 **OGRErr OSRSetCEA** ( **OGRSpatialReferenceH** *hSRS*, **double** *dfStdP1*, **double** *dfCentralMeridian*, **double** *dfFalseEasting*, **double** *dfFalseNorthing* )

Cylindrical Equal Area

13.19.2.73 **OGRErr OSRSetCompoundCS** ( **OGRSpatialReferenceH** *hSRS*, **const char \*** *pszName*, **OGRSpatialReferenceH** *hHorzSRS*, **OGRSpatialReferenceH** *hVertSRS* )

Setup a compound coordinate system.

This function is the same as **OGRSpatialReference::SetCompoundCS()** (p. ??)

13.19.2.74 **OGRErr OSRSetCS** ( **OGRSpatialReferenceH** *hSRS*, **double** *dfCenterLat*, **double** *dfCenterLong*, **double** *dfFalseEasting*, **double** *dfFalseNorthing* )

Cassini-Soldner

13.19.2.75 **OGRErr OSRSetEC** ( **OGRSpatialReferenceH** *hSRS*, **double** *dfStdP1*, **double** *dfStdP2*, **double** *dfCenterLat*, **double** *dfCenterLong*, **double** *dfFalseEasting*, **double** *dfFalseNorthing* )

Equidistant Conic

13.19.2.76 **OGRErr OSRSetEckert** ( **OGRSpatialReferenceH** *hSRS*, **int** *nVariation*, **double** *dfCentralMeridian*, **double** *dfFalseEasting*, **double** *dfFalseNorthing* )

Eckert I-VI

13.19.2.77 **OGRErr OSRSetEckertIV** ( **OGRSpatialReferenceH** *hSRS*, **double** *dfCentralMeridian*, **double** *dfFalseEasting*, **double** *dfFalseNorthing* )

Eckert IV

13.19.2.78 **OGRErr OSRSetEckertVI** ( **OGRSpatialReferenceH** *hSRS*, **double** *dfCentralMeridian*, **double** *dfFalseEasting*, **double** *dfFalseNorthing* )

Eckert VI

13.19.2.79 **OGRErr OSRSetEquirectangular** ( **OGRSpatialReferenceH** *hSRS*, **double** *dfCenterLat*, **double** *dfCenterLong*, **double** *dfFalseEasting*, **double** *dfFalseNorthing* )

Equirectangular

13.19.2.80 **OGRErr OSRSetEquirectangular2** ( **OGRSpatialReferenceH** *hSRS*, **double** *dfCenterLat*, **double** *dfCenterLong*, **double** *dfPseudoStdParallel1*, **double** *dfFalseEasting*, **double** *dfFalseNorthing* )

Equirectangular generalized form

13.19.2.81 **OGRErr CPL\_STDCALL OSRSetFromUserInput** ( **OGRSpatialReferenceH** *hSRS*, **const char \*** *pszDef* )

Set spatial reference from various text formats.

This function is the same as **OGRSpatialReference::SetFromUserInput()** (p. ??)

13.19.2.82 **OGRErr OSRSetGaussSchreiberTMercator** ( **OGRSpatialReferenceH** *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing* )

Gauss Schreiber Transverse Mercator

13.19.2.83 **OGRErr OSRSetGeocCS** ( **OGRSpatialReferenceH** *hSRS*, const char \* *pszName* )

Set the user visible PROJCS name.

This function is the same as **OGRSpatialReference::SetGeocCS()** (p. ??)

Since

OGR 1.9.0

13.19.2.84 **OGRErr OSRSetGeogCS** ( **OGRSpatialReferenceH** *hSRS*, const char \* *pszGeogName*, const char \* *pszDatumName*, const char \* *pszSpheroidName*, double *dfSemiMajor*, double *dfInvFlattening*, const char \* *pszPMName*, double *dfPMOffset*, const char \* *pszAngularUnits*, double *dfConvertToRadians* )

Set geographic coordinate system.

This function is the same as **OGRSpatialReference::SetGeogCS()** (p. ??)

13.19.2.85 **OGRErr OSRSetGEOS** ( **OGRSpatialReferenceH** *hSRS*, double *dfCentralMeridian*, double *dfSatelliteHeight*, double *dfFalseEasting*, double *dfFalseNorthing* )

GEOS - Geostationary Satellite View

13.19.2.86 **OGRErr OSRSetGH** ( **OGRSpatialReferenceH** *hSRS*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing* )

Goode Homolosine

13.19.2.87 **OGRErr OSRSetGnomonic** ( **OGRSpatialReferenceH** *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing* )

Gnomonic

13.19.2.88 **OGRErr OSRSetGS** ( **OGRSpatialReferenceH** *hSRS*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing* )

Gall Stereographic

13.19.2.89 **OGRErr OSRSetHOM** ( **OGRSpatialReferenceH** *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfAzimuth*, double *dfRectToSkew*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing* )

Set a Hotine Oblique Mercator projection using azimuth angle.

Hotine Oblique Mercator using azimuth angle

This is the same as the C++ method **OGRSpatialReference::SetHOM()** (p. ??)



13.19.2.90 `OGRErr OSRSetHOM2PNO ( OGRSpatialReferenceH hSRS, double dfCenterLat, double dfLat1, double dfLong1, double dfLat2, double dfLong2, double dfScale, double dfFalseEasting, double dfFalseNorthing )`

Set a Hotine Oblique Mercator projection using two points on projection centerline.

Hotine Oblique Mercator using two points on centerline

This is the same as the C++ method `OGRSpatialReference::SetHOM2PNO()` (p. ??)

13.19.2.91 `OGRErr OSRSetIGH ( OGRSpatialReferenceH hSRS )`

Interrupted Goode Homolosine

13.19.2.92 `OGRErr OSRSetWMPolyconic ( OGRSpatialReferenceH hSRS, double dfLat1, double dfLat2, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing )`

International Map of the World Polyconic

13.19.2.93 `OGRErr OSRSetKrovak ( OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfAzimuth, double dfPseudoStdParallelLat, double dfScale, double dfFalseEasting, double dfFalseNorthing )`

Krovak Oblique Conic Conformal

13.19.2.94 `OGRErr OSRSetLAEA ( OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing )`

Lambert Azimuthal Equal-Area

13.19.2.95 `OGRErr OSRSetLCC ( OGRSpatialReferenceH hSRS, double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing )`

Lambert Conformal Conic

13.19.2.96 `OGRErr OSRSetLCC1SP ( OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing )`

Lambert Conformal Conic 1SP

13.19.2.97 `OGRErr OSRSetLCCB ( OGRSpatialReferenceH hSRS, double dfStdP1, double dfStdP2, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing )`

Lambert Conformal Conic (Belgium)

13.19.2.98 `OGRErr OSRSetLinearUnits ( OGRSpatialReferenceH hSRS, const char * pszUnits, double dfInMeters )`

Set the linear units for the projection.

This function is the same as `OGRSpatialReference::SetLinearUnits()` (p. ??)

13.19.2.99 `OGRErr OSRSetLinearUnitsAndUpdateParameters ( OGRSpatialReferenceH hSRS, const char * pszUnits, double dfInMeters )`

Set the linear units for the projection.

This function is the same as **OGRSpatialReference::SetLinearUnitsAndUpdateParameters()** (p. ??)

13.19.2.100 OGRErr OSRSetLocalCS ( OGRSpatialReferenceH *hSRS*, const char \* *pszName* )

Set the user visible LOCAL\_CS name.

This function is the same as **OGRSpatialReference::SetLocalCS()** (p. ??)

13.19.2.101 OGRErr OSRSetMC ( OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing* )

Miller Cylindrical

13.19.2.102 OGRErr OSRSetMercator ( OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing* )

Mercator

13.19.2.103 OGRErr OSRSetMollweide ( OGRSpatialReferenceH *hSRS*, double *dfCentralMeridian*, double *dfFalseEasting*, double *dfFalseNorthing* )

Mollweide

13.19.2.104 OGRErr OSRSetNormProjParm ( OGRSpatialReferenceH *hSRS*, const char \* *pszParmName*, double *dfValue* )

Set a projection parameter with a normalized value.

This function is the same as **OGRSpatialReference::SetNormProjParm()** (p. ??)

13.19.2.105 OGRErr OSRSetNZMG ( OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing* )

New Zealand Map Grid

13.19.2.106 OGRErr OSRSetOM ( OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfAzimuth*, double *dfRectToSkew*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing* )

Oblique Mercator (aka HOM (variant B))

13.19.2.107 OGRErr OSRSetOrthographic ( OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing* )

Orthographic

13.19.2.108 OGRErr OSRSetOS ( OGRSpatialReferenceH *hSRS*, double *dfOriginLat*, double *dfCMeridian*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing* )

Oblique Stereographic

13.19.2.109 `OGRERR OSRSetPolyconic ( OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing )`

Polyconic

13.19.2.110 `OGRERR OSRSetProjCS ( OGRSpatialReferenceH hSRS, const char * pszName )`

Set the user visible PROJCS name.

This function is the same as `OGRSpatialReference::SetProjCS()` (p. ??)

13.19.2.111 `OGRERR OSRSetProjection ( OGRSpatialReferenceH hSRS, const char * pszProjection )`

Set a projection name.

This function is the same as `OGRSpatialReference::SetProjection()` (p. ??)

13.19.2.112 `OGRERR OSRSetProjParm ( OGRSpatialReferenceH hSRS, const char * pszParmName, double dfValue )`

Set a projection parameter value.

This function is the same as `OGRSpatialReference::SetProjParm()` (p. ??)

13.19.2.113 `OGRERR OSRSetPS ( OGRSpatialReferenceH hSRS, double dfCenterLat, double dfCenterLong, double dfScale, double dfFalseEasting, double dfFalseNorthing )`

Polar Stereographic

13.19.2.114 `OGRERR OSRSetRobinson ( OGRSpatialReferenceH hSRS, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing )`

Robinson

13.19.2.115 `OGRERR OSRSetSinusoidal ( OGRSpatialReferenceH hSRS, double dfCenterLong, double dfFalseEasting, double dfFalseNorthing )`

Sinusoidal

13.19.2.116 `OGRERR OSRSetSOC ( OGRSpatialReferenceH hSRS, double dfLatitudeOfOrigin, double dfCentralMeridian, double dfFalseEasting, double dfFalseNorthing )`

Swiss Oblique Cylindrical

13.19.2.117 `OGRERR OSRSetStatePlane ( OGRSpatialReferenceH hSRS, int nZone, int bNAD83 )`

Set State Plane projection definition.

This function is the same as `OGRSpatialReference::SetStatePlane()` (p. ??).

13.19.2.118 `OGRERR OSRSetStatePlaneWithUnits ( OGRSpatialReferenceH hSRS, int nZone, int bNAD83, const char * pszOverrideUnitName, double dfOverrideUnit )`

Set State Plane projection definition.

This function is the same as **OGRSpatialReference::SetStatePlane()** (p. ??).

13.19.2.119 OGRErr OSRSetStereographic ( OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing* )

Stereographic

13.19.2.120 OGRErr OSRSetTargetLinearUnits ( OGRSpatialReferenceH *hSRS*, const char \* *pszTargetKey*, const char \* *pszUnits*, double *dfInMeters* )

Set the linear units for the target node.

This function is the same as **OGRSpatialReference::SetTargetLinearUnits()** (p. ??)

Since

OGR 1.9.0

13.19.2.121 OGRErr OSRSetTM ( OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing* )

Transverse Mercator

Special processing available for Transverse Mercator with GDAL >= 1.10 and PROJ >= 4.8 : see **OGRSpatialReference::exportToProj4()** (p. ??).

13.19.2.122 OGRErr OSRSetTMG ( OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing* )

Tunesia Mining Grid

13.19.2.123 OGRErr OSRSetTMSO ( OGRSpatialReferenceH *hSRS*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing* )

Transverse Mercator (South Oriented)

13.19.2.124 OGRErr OSRSetTMVariant ( OGRSpatialReferenceH *hSRS*, const char \* *pszVariantName*, double *dfCenterLat*, double *dfCenterLong*, double *dfScale*, double *dfFalseEasting*, double *dfFalseNorthing* )

Transverse Mercator variant

13.19.2.125 OGRErr OSRSetTOWGS84 ( OGRSpatialReferenceH *hSRS*, double *dfDX*, double *dfDY*, double *dfDZ*, double *dfEX*, double *dfEY*, double *dfEZ*, double *dfPPM* )

Set the Bursa-Wolf conversion to WGS84.

This function is the same as **OGRSpatialReference::SetTOWGS84()** (p. ??).

13.19.2.126 OGRErr OSRSetUTM ( OGRSpatialReferenceH *hSRS*, int *nZone*, int *bNorth* )

Set UTM projection definition.

This is the same as the C++ method **OGRSpatialReference::SetUTM()** (p. ??)

13.19.2.127 **OGR**Err OSRSetVDG ( **OGR**SpatialReferenceH *hSRS*, double *dfCenterLong*, double *dfFalseEasting*, double *dfFalseNorthing* )

VanDerGrinten

13.19.2.128 **OGR**Err OSRSetVertCS ( **OGR**SpatialReferenceH *hSRS*, const char \* *pszVertCSName*, const char \* *pszVertDatumName*, int *nVertDatumType* )

Setup the vertical coordinate system.

This function is the same as **OGRSpatialReference::SetVertCS()** (p. ??)

Since

OGR 1.9.0

13.19.2.129 **OGR**Err OSRSetWagner ( **OGR**SpatialReferenceH *hSRS*, int *nVariation*, double *dfFalseEasting*, double *dfFalseNorthing* )

Wagner I – VII

13.19.2.130 **OGR**Err OSRSetWellKnownGeogCS ( **OGR**SpatialReferenceH *hSRS*, const char \* *pszName* )

Set a GeogCS based on well known name.

This function is the same as **OGRSpatialReference::SetWellKnownGeogCS()** (p. ??)

13.19.2.131 **OGR**Err OSRStripCTParms ( **OGR**SpatialReferenceH *hSRS* )

Strip OGC CT Parameters.

This function is the same as **OGRSpatialReference::StripCTParms()** (p. ??).

13.19.2.132 **OGR**Err OSRValidate ( **OGR**SpatialReferenceH *hSRS* )

Validate SRS tokens.

This function is the same as the C++ method **OGRSpatialReference::Validate()** (p. ??).

## 13.20 ogrsf\_frmts.h File Reference

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

### Classes

- class **OGR**Layer
- class **OGR**DataSource
- class **OGR**SFDriver
- class **OGR**SFDriverRegistrar

## Functions

- void **OGRRegisterAll** ()  
*Register all drivers.*

### 13.20.1 Detailed Description

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